A Review of Spiking Neuron Models and Applications

by

Roberto Coelho de Berrêdo

B.S., Pontifical Catholic University of Minas Gerais, Brazil, 1978.

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in the

Post-Graduation Program in Electrical Engineering

we accept this dissertation as conforming
to the required standard

Universidade Federal de Minas Gerais

November 2005

Abstract

Although the nervous system and nervous cells have been studied since ancient times, almost all the important advances on the knowledge base took place along the past century, when neuroscience evolved and developed more complex and detailed neuron models. However, engineering applications were – and still are – practically limited to the basic neuron models, like the MCP [MP43] and its variations. In traditional artificial neural networks, the neuron behavior is described only in terms of firing rate, while most real neurons, commonly known as spiking neurons, transmit information by pulses, also called action potentials or spikes. From these considerations a major question raises immediately: if we were able to build powerfull applications in all fields of engineering using these simple models, what would it be possible to do with the more complex models? This question is the fundamental motivation of the present work.

Given the importance of more realistic neuron models, our main objective is to present a general and comprehensive overview of spiking neurons, ranging from biological neuron features to examples of practical applications. It will be also demonstrated how analog information can be coded by precisely timed spikes, emitted by different neurons, and how this coded information can be processed to produce usefull results [Maa97]. The aim of the present work is therefore to highlight what we believe will be one of the main components of the future computing machines: the spiking neuron.

As a contribution, besides the review itself, we present also a novel approach to the spiking neuron network architecture used in practical applications [Ruf98, BPK02], replacing the multiple synapse construction with single synapses, and thus enhancing the computational efficiency of the network.

Contents

List of Tables

List of Figures

To my family, professors and friends, for their support and patience.

Chapter 1

Introduction

1.1 Background

Although the nervous system and nervous cells have been studied since ancient times, almost all the important advances on the knowledge base took place along the past century, and were founded on five experimental disciplines: anatomy, embryology, physiology, pharmacology, and psychology [KSJ00]. Then, beginning in the middle of the past century, engineering was added as a sixth discipline and, reciprocally, neuroscience was also adopted by engineering, ensuing all the development of computational intelligence and making this subject a rather interdisciplinary one.

However, as neuroscience evolved and developed more complex and detailed neuron models, engineering applications were – and still are – practically limited to the basic neuron models, like the MCP [MP43] and its variations. In traditional artificial neural networks, the neuron behavior is described only in terms of firing rate, while most real neurons, commonly known as spiking neurons, transmit information by pulses, also called action potentials or spikes. From these considerations a major question raises immediately: if we were able to build powerfull applications in all fields of engineering using these simple models, what would it be possible to do with the more complex models?

Moreover, in the last ten years or so all the main questions formerly addressed by neuroscientists only to cellular biology began to be effectively explored in the molecular level. The research in molecular biology contributed to the knowledge about ion channels and receptors, two impor-

tant elements in neural signaling, making it possible to describe the first molecular structure of a ionic channel [KSJ00]. This enhanced capacity of neuronal modelling came to shed some light into questions like: How do the nervous cells communicate among them? How is this communication modified by experience? How different interconnection patterns originate different perceptions and motor actions?

From the engineering point of view, it is clear that the answers for all these questions will only be possible with a deeper comprehension of the biological neuron and how they can do fast and reliable computation [MB98]. This apparent difficulty should rather be regarded as an opportunity to use spike-timing as an additional variable in the information processing by neural networks [Boh04]. Although much progress has been achieved in the last two decades, there are a few fundamental questions still not completely solved, like how do real neurons transmit information or how to use the spike timing efficiently to process information [Nat96, GK02b].

As a matter of fact, realistic and complex neural models have been used since long to simulate real networks, but always with emphasis on the spike rate. It was also only in the last ten years, however, that we have witnessed a shift on the emphasis in the artificial neural network community toward practical spiking neural networks – SNN [TH86, Hop88, Hop95, Nat96]. This shift was motivated by recent biological discoveries and by the increasing available computer power, so that many researchers began considering pulse-coupled neural networks with spike-timing as an essential component in information processing by neural networks [Maa97, Ger01], artificial or not.

Independent of the choice of the point of view (molecular, cellular, systemic, behavioral, or cognitive), the human nervous system is a masterpiece of biological machinery, worth to be better studied and employed as a powerfull engineering tool. All its accomplishments are more than sufficient motivation for willing to understand how each of its many components works. Not to mention that the debilitating and costly effects of neurological and psychiatric disease add a further sense of urgency to this quest [PAF+04].

It is likely that next great change in computer science and information technology may come

from mimicking the way by which biological neural systems process information. To accomplish this, engineers and computer scientists will have draw on expertise in subjects not usually associated with their fields, including organic chemistry, molecular biology, bioengineering, and smart materials. In a not so distant future we will be possibly using proteins and other molecules for information-processing, molecular recognition, computation in nonlinear media, computers based on physical reaction-diffusion systems found in chemical media, DNA computing, bioelectronics and protein-based optical computing, and biosensors [SARC03].

1.2 Objective and Organization of the Work

Given the importance of more realistic neuron models, as exposed in the previous section, the main objective of this work is to present a general and comprehensive overview of spiking neurons, ranging from biological neuron features to examples of practical applications. It will be also demonstrated how analog information can be coded by precisely timed spikes, emitted by different neurons, and how this coded information can be processed to produce usefull results, as demonstrated theoretically in [Maa97]. The aim of the present work is therefore to highlight what we believe will be one of the main components of the future computing machines: the spiking neuron. In this work we intend to make a review on spiking neurons, presenting a comprehensive scenario of the foundations and a few application examples as well.

The present knowledge of the nervous system has reached an enormous level of detail, making it impossible even to summarize its main components in only one chapter. Therefore, in **Chapter 2** we limited ourselves to mention only those main components with a brief description of its basic properties, including ion channels, synapses, dendrites and axons, as well as the electrical features like excitatory (inhibitory) postsynaptic potential and the spike. The purpose of this chapter is to provide enough information about the nervous system components, preparing a basis for the next chapter, when we will discuss the features and characteristics we must take into account when designing a neuron model. For the sake of objectiveness we do not included those components with less significant functions in the information processing in a nervous system (e.g. the glial cells), although they may play a significant role in the brain structure.

Since the works of Santiago Ramon y Cajal [yC03] and Camillo Golgi, a vast number of theoretical neuron models have been created, with a modern phase beginning with the work of Hodgkin and Huxley [HH52]. All these models range from special purpose to generic models, aiming to emulate one or various features of the nervous systems, most of which are mentioned in the previous chapter. In **Chapter 3** we concisely present some of the most used theoretical neuron models, from the simple ones, like the integrate-and-fire, to the more complex ones, like the compartmental models. In real nervous systems there is a wide range of neuron types, each one assigned to do a specific function, making it virtually impossible to create a model that meets all the requirements.

Historically, much of the research effort to comprehend the neural mechanisms involved in information processing in the brain has been spent with neuronal circuits and synaptic organization, largely neglecting the electrophysical properties of the neurons. In **Chapter 4**, some of the theoretical models presented in Chapter 3 are implemented, giving some examples of the application of the models to simulate real neurons. These examples demonstrate the existence of a relationship between electrophysiology, bifurcations, and computational properties of neurons, showing also the foundations of the dynamical behavior of neural systems.

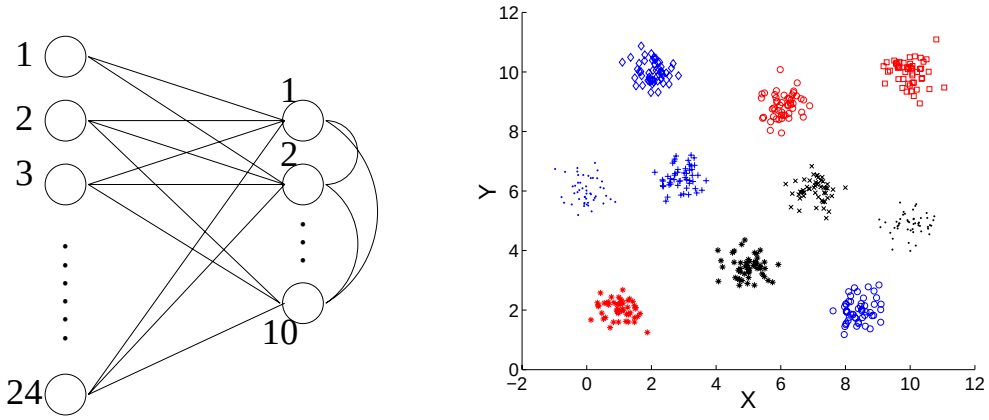

Figure 1.1: The SNN at left was capable of correctly separate ten clusters. The lateral connections linking the output neurons are strong inhibitory synapses, disabling all other neurons to fire after the first neuron has fired, thus implementing the winner-takes-all process. Each dimension of the input was coded by a receptive field with 12 neurons.

In **Chapter 5** we present an example of a practical application using spiking neurons and

temporal coding to process information, building a spiking neural network – SNN to perform a clustering task, as shown in figure 1.1. The input is encoded by means of receptive fields, as described in [Ruf98] and [BPK02]. The delay and weight adaptation uses a multiple syanapse approach, dividing each synapse into subsynapses, each one with a different fixed delay. The delay selection is then performed by a Hebbian reinforcement learning algorithm, also keepping ressemblance with biological neural networks. As a contribution of this work, we propose in this chapter an alternative architecture to do the same task with less computing effort. This new approach is based on the use of an EPSP produced by a single synapse that is equivalent to the EPSP produced by the multiple synapse architecture of the original method.

Finally, in Appendix A we listed some of the Matlab scripts [HL03] used in the development of Chapter 5, and in Appendix B we listed some interesting internet sites, which were very helpfull to acomplish this work and where a lot of usefull material can be found.

Chapter 2

Biological Neurons and Neuronal Features

2.1 Introduction

The aim of this chapter is to present some elementary notions of neuroscience, beginning with a generic phenomenological model of the biological neuron and its basic concepts, like action potentials, postsynaptic potentials, firing thresholds and refractoriness. This will serve as the starting point to the following chapter, where the various theoretical models used in computational neuroscience will be discussed.

The human brain is made of hundreds of billions of cells and a full description of this system, as far as it has been uncovered, would take several volumes of written material. Due to this high complexity, it is beyond the scope of this work to present a complete introduction to this subject. Therefore, we limit ourselves to a highly selective and simplistic biological background to provide the minimum amount of information necessary to the development of this work.

Even the basic nervous cell, the neuron, has many different forms and is spread over several anatomically different structures, like brain-stem, cerebellum, and cortex. In the following sections we limit ourselves to the presentation of the basic concepts of a generic neuron model and the most important types of behaviors and features of spiking neurons.

2.2 A Generic Spiking Neuron

Throughout the past hundred years, researchers have been uncovering an increasingly complex brain structure. The elementary processing units in the brain are the neurons, which are connected to each other in an intricate pattern and can occur in many shapes and sizes. As stated above, the neuron described here is very simplified and its purpose is to serve as a 'template' neuron for further development. Our generic neuron has four functionally distinct parts, called dendritic tree, soma, axon and synapse, like the diagram shown in figure 2.1 [yC03]. Roughly speaking, signals from other neurons are collected by the dendrites (input device) and are transmitted to the soma (central processing unit). If the total excitation caused by the input is sufficient, i.e., above a threshold, an output signal (action potential, or spike) is emitted and propagated along the axon (output device) and its branches to other neurons. It is in the transition zone between the soma and the axon, the axon hillock, where the the essential non-linear processing step occurs [KSJ00].

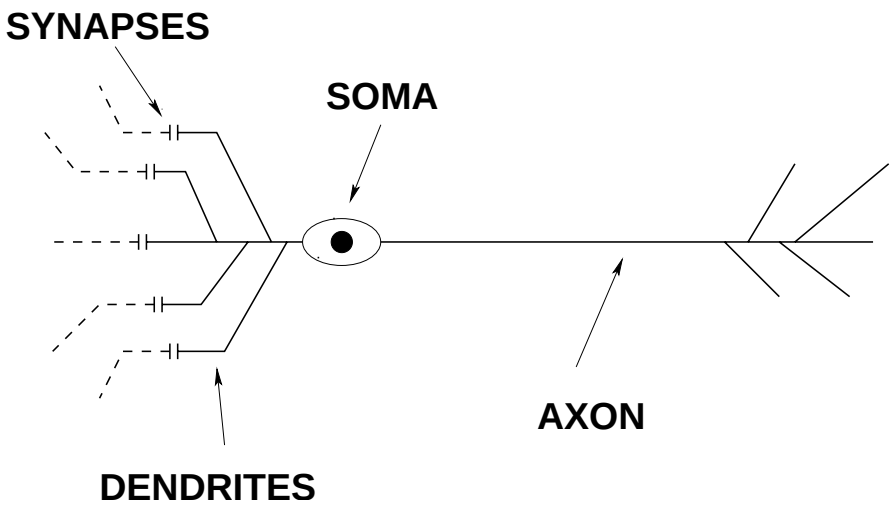

Figure 2.1: A simplified generic neuron diagram (based on [yC03]).

It is common to refer to a sending neuron as the presynaptic neuron and to the receiving neuron as a postsynaptic neuron. A neuron in vertebrate cortex often makes connections to more than 10^4 postsynaptic neurons [RWdRvSB97]. Many of its axonal branches end in the direct neighborhood of the neuron, but the axon can also stretch over several millimeters and connect to neurons in other areas of the brain.

The neuronal signals can be observed by placing a fine electrode close to the soma or axon of a neuron. The voltage trace in a typical recording shows a sequence of short pulses, called action potentials or spikes. A chain of pulses emitted by a single neuron is usually called a spike train - a sequence of stereotyped events which occur at regular or irregular intervals. The duration of an action potential is typically in the range of $1 - 2ms$ with an amplitude of about $100mV$ [GK02b, PAF+04], as showed in figure 2.3. Since all spikes of a given neuron look alike, the form of the action potential does not carry any information. Rather, it is the number and the timing of spikes which matter [MB98].

2.2.1 The Nervous Cell Membrane

Neurons employ several different types of electrical signals to encode and transfer information. These signals are caused by responses to stimuli and are generated by means of elaborated mechanisms, usually based on the flow of ions across the plasma membranes of nervous cells [KSJ00]. The electrical potential across the membrane, called membrane potential, is generated by different concentrations of specific ions inside and outside of nerve cells. Each membrane has its specific resting potential (typically -40 to -90mV) [PAF+04]. The ion concentration (and the membrane potential) gradient is regulated by the active transporters and ion channels.

The active transporters, also known as ion pumps, are proteins that actively move ions into or out of cells against their concentration gradients. The ion channels are proteins that allow only certain kinds of ions to cross the membrane in the direction of their ion concentration gradients, and thus characterizing the selective permeability of the membrane to each type of ion. These two mechanisms basically work against each other, generating membrane resting potentials, action potentials and synaptic potentials.

2.2.2 Synapses

Information from one neuron flows to another neuron across a small gap, called synapse, separating an axonal branch and the dendrite (or the soma) of a receiving neuron. The most common type of synapse in the vertebrate brain is the chemical synapse [PAF+04, KSJ00], which act through

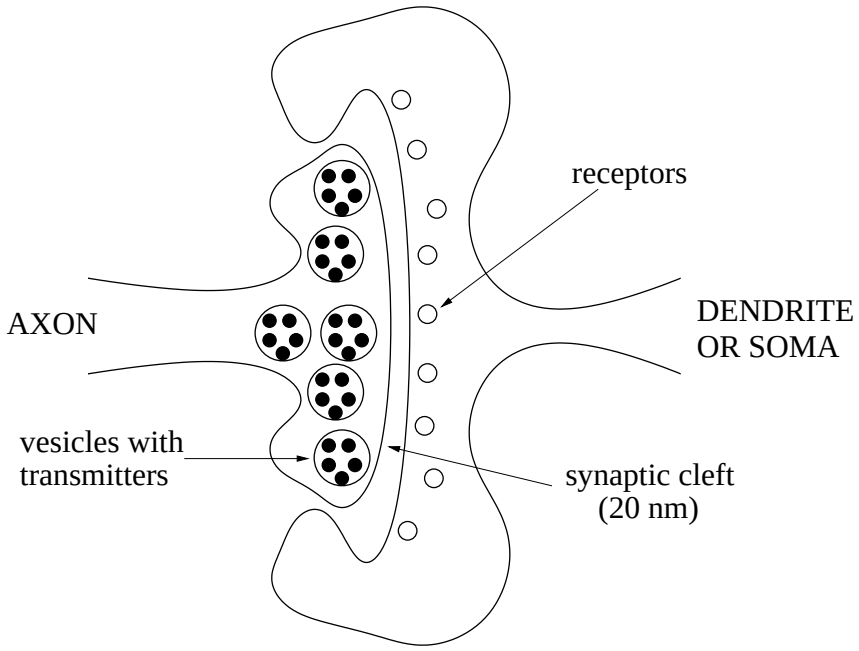

Figure 2.2: Diagram of a synapse with its vesicles, neurotransmitters and neuroreceptors.

biochemical processes, neurotransmitters and receptors to change the membrane potential of the postsynaptic neuron. In chemical synapses, an action potential cannot cross the synaptic cleft between neurons. Instead, the nerve impulse is carried by chemicals called neurotransmitters, which are made by the cell that is sending the impulse (the presynaptic neuron) and stored in synaptic vesicles at the end of the axon. The cell that is receiving the nerve impulse (the post-synaptic neuron) has chemical-gated ion channels in its membrane, called neuroreceptors. These have specific binding sites for the neurotransmitters. As shown in figure 2.2 chemical synapse consists of [PAF+04]:

- a presynaptic ending that contains neurotransmitters, mitochondria, and other cell organelles (where neurotransmitters are synthesized),
- a postsynaptic ending that contains receptor sites for neurotransmitters and has neuroreceptors in the membrane,
- a synaptic cleft or space between the presynaptic and postsynaptic endings. It is about 20nm wide.

The action potential transmission process can be described by the following steps [KSJ00]:

a. Transmitter is synthesized and then stored in vesicles.

b. An action potential reaches the presynaptic terminal.

c. At the end of the presynaptic neuron there are voltage-gated calcium channels. When an action potential reaches the synapse, depolarizing the presynaptic terminal, these channels open, causing calcium ions (Ca^{2+}) to flow into the cell.

d. These calcium ions (Ca^{2+}) cause the synaptic vesicles to fuse with the cell membrane, releasing their contents (the neurotransmitter chemicals) via exocytosis.

e. The neurotransmitters are released into the synaptic cleft and diffuse to the postsynaptic terminal across the synaptic cleft.

f. The neurotransmitter binds to the neuroreceptors in the postsynaptic membrane, causing its channels to open (or close). This causes a depolarization (EPSP) or hyperpolarization (IPSP) of the postsynaptic cell membrane, which may initiate an action potential, if the threshold is reached. See figure 2.5.

g. Retrival of vesicular membrane from plasma membrane. The neurotransmitter is broken down by a specific enzyme in the synaptic cleft (e.g., the enzyme acetylcholinesterase breaks down the neurotransmitter acetylcholine). The breakdown products are absorbed by the presynaptic neuron by endocytosis and used to re-synthesize more neurotransmitters, using energy from the mitochondria. This stops the synapse being permanently on.

Apart from chemical synapses, neurons in the mammalian nervous systems can also be coupled by other different types of synapses. The human nervous system uses a number of different neurotransmitters and neuroreceptors, and they do not all work in the same way. We can group synapses into 5 types [PAF+04, KSJ00]:

Excitatory Ion Channel Synapses: These synapses have neuroreceptors that are sodium channels. When the channels open, positive ions flow in, causing a local depolarization and making an action potential more likely. This was the kind of synapse described above. Typical neurotransmitters are acetylcholine, glutamate or aspartate. Neuroreceptors are Na+ channels. When Na^{+} channels open, local depolarization occurs, if threshold is reached, then action potential is initiated.

Inhibitory Ion Channel Synapses: These synapses have neuroreceptors that are chloride channels. When the channels open, negative ions Cl^- flow in causing a local hyperpolarization and making an action potential less likely. So, with these synapses an impulse in one neuron can inhibit an impulse in the next. Typical neurotransmitters are glycine or GABA (γ-aminobutyric acid).

Non Channel Synapses: These synapses have neuroreceptors that are not channels at all, but instead are membrane-bound enzymes. When activated by the neurotransmitter, they catalyze the production of a 'chemical messenger' inside the cell, which, in turn, can affect many aspects of the cell's metabolism. In particular, they can alter the number and sensitivity of the ion channel receptors in the same cell. These synapses are involved in slow and long-lasting responses like learning and memory. Typical neurotransmitters are adrenaline, noradrenaline (NB adrenaline is also called epinephrine), dopamine, serotonin, endorphin, angiotensin, and acetylcholine.

Neuromuscular Junctions: These are the synapses formed between motor neurons and muscle cells. They always use the neurotransmitter acetylcholine, and are always excitatory. Motor neurons also form specialized synapses with secretory cells. neuroreceptors are membrane-bound enzymes. When activated, they catalyze the 'chemical messenger', which in turn can affect the sensitivity of the ion channel receptors in the cell synapses formed between motor neurons and muscle cells. They always use the neurotransmitter acetylcholine, and are always excitatory.

Electrical Synapses: In these synapses, the membranes of the two cells actually touch, and they share proteins. This allows the action potential to pass directly from one membrane to the next. They are very fast, but are quite rare, found only in the heart and in the eye.

2.2.3 The Postsynaptic Potential – PSP

The potential difference between the interior of the cell (soma) and its surroundings is called the membrane potential. This potential is directly affected by the postsynaptic potentials - PSPs generated by the spikes received from presynaptic neurons. If the membrane potential reaches a threshold, an action potential (spike) is triggered and sent out through the axon and its branches to the postsynaptic neurons. If the postsynaptic potential is positive, it is said to be excitatory (EPSP)

and if the change is negative, the synapse is inhibitory (IPSP). See figure 2.3. The process of spike transmission through the axon has an associated delay, called the axonal delay [PAF+04, KSJ00].

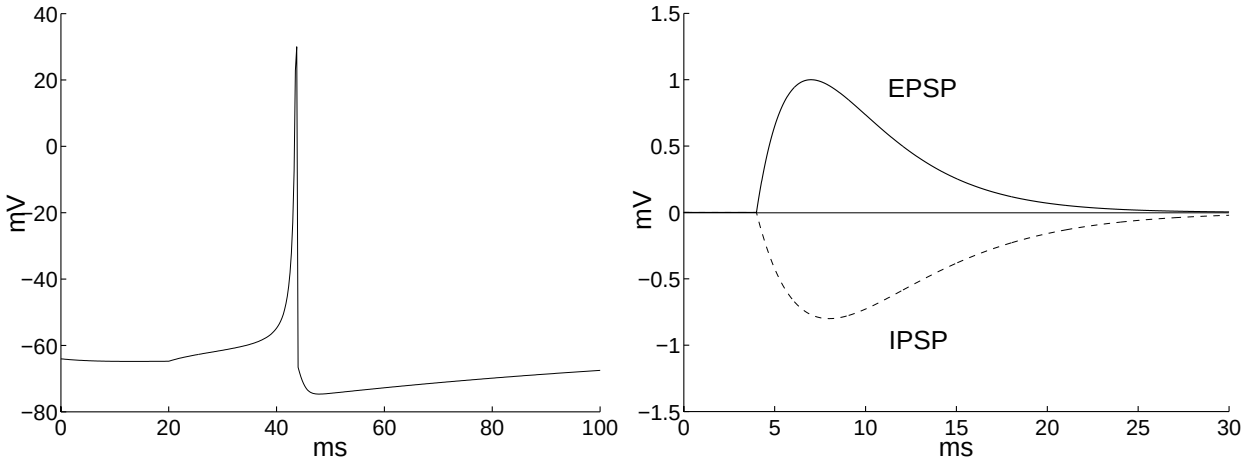

Figure 2.3: Examples of action potential (left) and EPSP and IPSP (right).

At rest, the cell membrane already has a strong negative polarization of about $-65mV$ [Wil99]. An input at an excitatory synapse reduces the negative polarization of the membrane and is therefore called depolarizing. Likewise, an inhibitory synapse increases the negative polarization of the membrane even further and is called hyperpolarizing. Single EPSPs have amplitudes in the range of $1mV$ and the threshold for spike initiation is about $20-30mV$ [GK02b] above the resting potential.

At this point we need to define some notations and symbols that will be used throughout this work. A presynaptic neuron is denoted by i and a postsynaptic neuron by j. The set of all presynaptic neurons to neuron j is then defined as $\Gamma_j = \{i | i \text{ presynaptic to } j\}$. Similarly, the set of all postsynaptic neurons to the neuron i is defined as $\Gamma_i = \{j | j \text{ postsynaptic to } i\}$.

The moment when a given neuron j emits an action potential will be referred to as the firing time of that neuron and will be denoted by $t_j^{(f)}$, with f varying from 1 to n. The spike train of neuron j is then characterized by the set of firing times given by equation (2.1).

$$F_j = \{t_j^{(1)}, t_j^{(2)}, ..., t_j^{(n)}\} \tag{2.1}$$

where $t_j^{(n)}$ is the most recent spike of neuron j, also denoted by $\hat{t}$.

The PSP is the response generated by a spike from the presynaptic neuron i in the postsynaptic neurons j and has the form shown in figure 2.3. It is important to note that, due to interneural distances and finite axonal transmission times, there may be a delay between the emission of a spike an the beginning of its corresponding PSP. Equation 2.2 gives a mathematical formulation for a generic PSP originated in neuron j by a spike emitted by neuron i [GK02b].

$$\epsilon_{ij}(t) = \left[exp\left(-\frac{t - t_i^{(f)} - \Delta_{ij}^{ax}}{\tau_m} \right) - exp\left(-\frac{t - t_i^{(f)} - \Delta_{ij}^{ax}}{\tau_s} \right) \right] \mathcal{H}(t - t_i^{(f)} - \Delta_{ij}^{ax}) \tag{2.2}$$

or in a more simple form [Boh03]

$$\epsilon_{ij}(t) = \frac{t - t_i^{(f)} - \Delta_{ij}^{ax}}{\tau} exp\left(1 - \frac{t - t_i^{(f)} - \Delta_{ij}^{ax}}{\tau} \right) \mathcal{H}(t - t_i^{(f)} - \Delta_{ij}^{ax}) \tag{2.3}$$

where τ_m, τ_s and τ are time constants, Δ_{ij}^{ax} is the axonal transmission delay, and $\mathcal{H}(\cdot)$ is the Heaviside step function. Figure 2.4 is an example of EPSP equation (2.3). Despite both equations have been widely used to simulate PSP, in a more formal and biological approach, the kernel ϵ_{ij} should be also a function of the last firing time of j, since the PSP depends on the state of the membrane potential, which, in turn, depends on $\hat{t}_j$. The kernel $\epsilon_{ij}(t - \hat{t}_j\ , t - t_i^{(f)} - \Delta_{ij}^{ax})$ can be interpreted as the time course of a postsynaptic potential evoked by the firing of a presynaptic neuron i at time $t_i^{(f)}$.

2.2.4 The Spike After Potential – SAP

Immediately after the emission of a spike, the membrane potential does not directly return to the resting potential u_{rest}, but passes through a phase of high hyperpolarization below the resting value [GK02b]. While the action potential is quickly rising or steeply falling, emission of a further spike is impossible. This effect is called absolute refractoriness. The spike after potential (SAP) being negative, makes the emission of a second spike immediately after the first pulse more difficult. This period of reduced sensitivity after a spike is called the relative refractory period. Figure 2.5 shows the membrane potential variation before, during and after the emission of a spike.

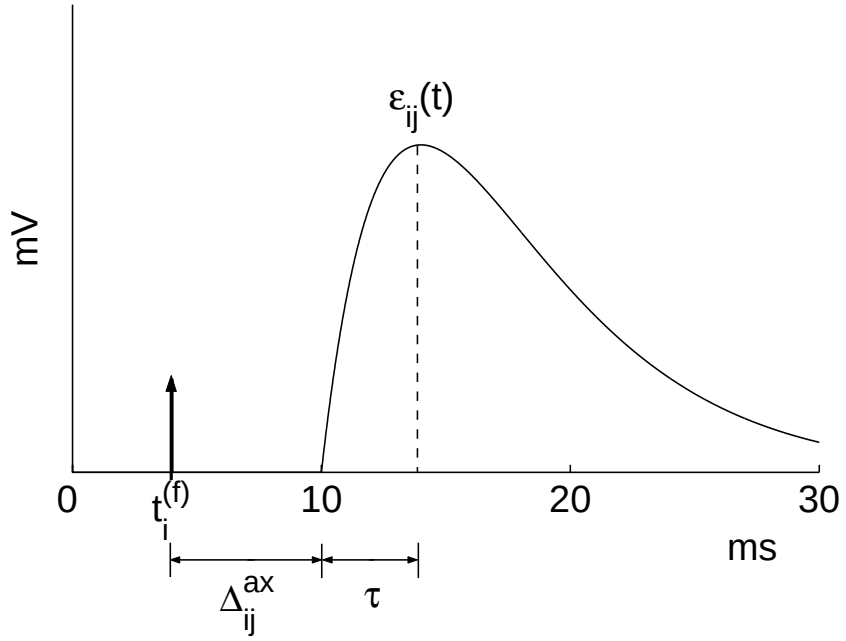

Figure 2.4: Example of an EPSP using equation 2.3, with $t_i^{(f)} = 4ms$, $\Delta_{ij}^{ax} = 6ms$ and $\tau = 4ms$.

2.3 The Equilibrium Potential

The neuron membrane is a nearly perfect electrical insulator and separates the intracellular environment from the surrounding environment. Embedded in the membrane are proteins that act as transporters and ion channels, actively transporting ions from one side to another, resulting in different ion concentrations in each side. This difference in the ion density generates a difference in the electrical potential between both sides of the membrane. At equilibrium, the voltage generated by the different density is given by the *Nernst equation* [Wil99].

$$E = \frac{kT}{q} \ln \left(\frac{C_{in}}{C_{out}} \right) \tag{2.4}$$

where E is the *equilibrium potential* (also called *Nernst potential*), k is the Boltzmann constant, T is the temperature in Kelvin degrees, q is the charge of the ion, and C_{in} and C_{out} are the respective ion concentrations inside and outside the membrane.

For example, the ion concentration of potassium (K^+) is higher inside the cell ($\approx 400mM$) than

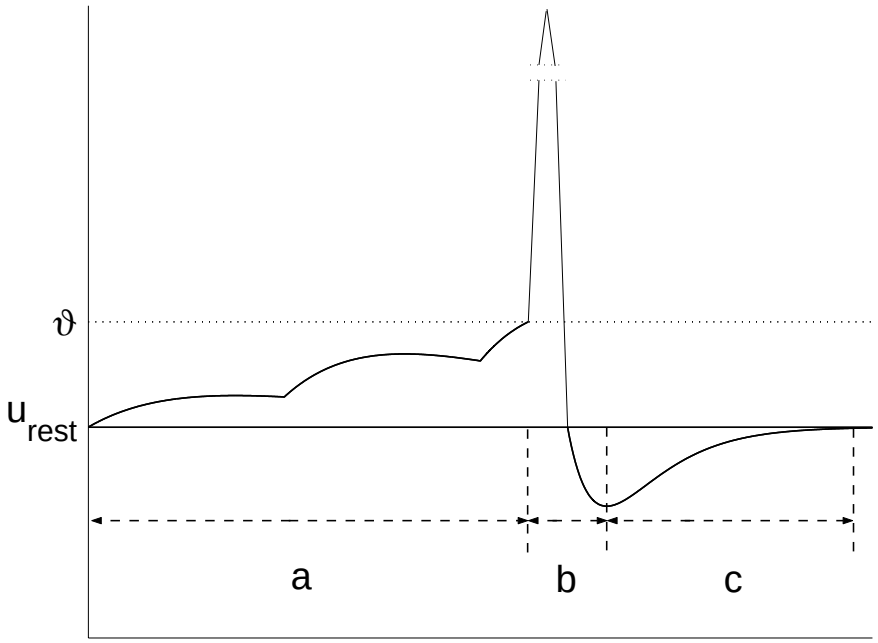

Figure 2.5: a- The membrane potential increases from the rest potential u_{rest} and reaches the threshold ϑ, b- The absolute refractory period: a spike is generated making impossible the emission of another spike, c- The relative refractory period: the hyperpolarization decreases and the membrane potential reaches the rest potential again.

in the extracellular liquid ($\approx 20mM$). Potassium ions have a positive charge $q = 1.6 \times 10^{-19}$ C. Application of the Nernst equation with the Boltzmann constant $k = 1.4 \times 10^{23}$ J/K yields $E_K = -77mV$ at room temperature. Similarly, the equilibrium potential for sodium ions (Na^+) is $E_{Na} = +55mV$ [GK02b]. The equilibrium potential is also called *reversal potential* because it represents the point where the ions flow reverses its direction to maintain the equilibrium.

In real cells, various ion types are simultaneously present and contribute to the voltage across the membrane. The resting potential u_{rest} is therefore determined by the dynamic equilibrium between the ion flow through the channels (permeability of the membrane) and active ion transport by the ion pumps. When there are more than one permeant ion, equation (2.4) is replaced by equation (2.5), which takes into account the permeability of the membrane to each ion of interest [PAF+04, KSJ00].

$$E = 58 \log \frac{P_K[K]_2 + P_Na[Na]_2 + P_Cl[Cl]_1}{P_K[K]_1 + P_Na[Na]_1 + P_Cl[Cl]_2} \tag{2.5}$$

In equation (2.5), P indicates the permeability of the membrane and the indexes 1 and 2 indicate the ion concentration in- and outside the membrane, respectively. The index of the Cl^- ion is inverted because the q factor of the Nernst equation was eliminated. The value 58 is for room temperature. Table 2.3 shows the intracellular concentrations of the main ions.

Table 2.1: Extracellular and intracellular ion concentrations for the main ions.

	Concentration (mM)	
Ion	*Intracellular*	*Extracellular*
Squid neuron		
Potassium (K^+)	400	20
Sodium (Na^+)	50	440
Chloride (Cl^-)	40-150	560
Calcium (Ca^2+)	0.0001	10
Mammalian neuron		
Potassium (K^+)	140	5
Sodium (Na^+)	5-15	145
Chloride (Cl^-)	4-30	110
Calcium (Ca^2+)	0.0001	1-2

2.4 Neurons Main Characteristics and Features

A last and major step before we can discuss models of spiking neurons is to define the neuro-computational features of real neurons that must be taken into account. In this section we make a brief review of 20 types of real (cortical) neurons response, considering the injection of simple dc pulses [Izh04].

a. **Tonic Spiking:** The neuron fires a spike train as long as the input current is on. This kind of behavior can be observed in the three types of cortical neurons: regular spiking excitatory neurons (RS), low-threshold spiking neurons (LTS) and fast spiking inhibitory neurons (FS).

b. **Phasic Spiking:** The neuron fires only a single spike at the onset of the input.

c. **Tonic Bursting:** The neuron fires periodic bursts of spikes when stimulated. This behavior

may be found in chattering neurons in cat neocortex. Since the frequency of bursts may be as high as $50Hz$, it is believed that such kind of neuron contributes to the gamma-frequency oscillations in the brain.

d. **Phasic Bursting:** The neuron fires only a single burst at the onset of the input.

e. **Mixed Model:** The neuron fires a phasic burst at the onset of stimulation and then switch to the tonic spiking mode. The intrinsically bursting excitatory neurons in mammalian neocortex may exhibit this behavior.

f. **Spike Frequency Adaptation:** The neuron fires tonic spikes with decreasing frequency. RS neurons usually exhibit adaptation of the interspike intervals, when these intervals increase until a steady state of periodic firing is reached, while FS neurons show no adaptation [GK02b].

g. **Class 1 Excitability:** Here, the frequency of tonic spiking depends on the strength of the input, i.e., the weaker is the dc current, the lower will be the spike frequency. The frequency of tonic spiking of neocortical neurons may vary form 2 to $200MHz$.

h. **Class 2 Excitability:** The neuron is unable to fire low-frequency spike trains and will either fire a high frequency spike train or will not fire at all.

i. **Spike Latency:** The neuron fires with a delay that depends on the strength of the stimulation. RS neurons in mammalian cortex can have latencies of tens of milliseconds.

j. **Subthreshold Oscillations:** The neuron exhibits oscillatory potentials.

k. **Frequency Preference and Resonance:** Due to the *resonance phenomenon*, a neuron having oscillatory potentials can respond selectively to the inputs having frequency content similar to the frequency of subthreshold oscillations. Such neurons are called *resonators* [Izh01].

l. **Integration and Coincidence Detection:** The neuron without oscillatory potentials acts as an integrator [Izh01]. The higher the frequency, the more likely it fires.

m. **Rebound Spike:** The neuron fires a post-inhibitory (rebound) spike when it receives and then is released from an inhibitory input.

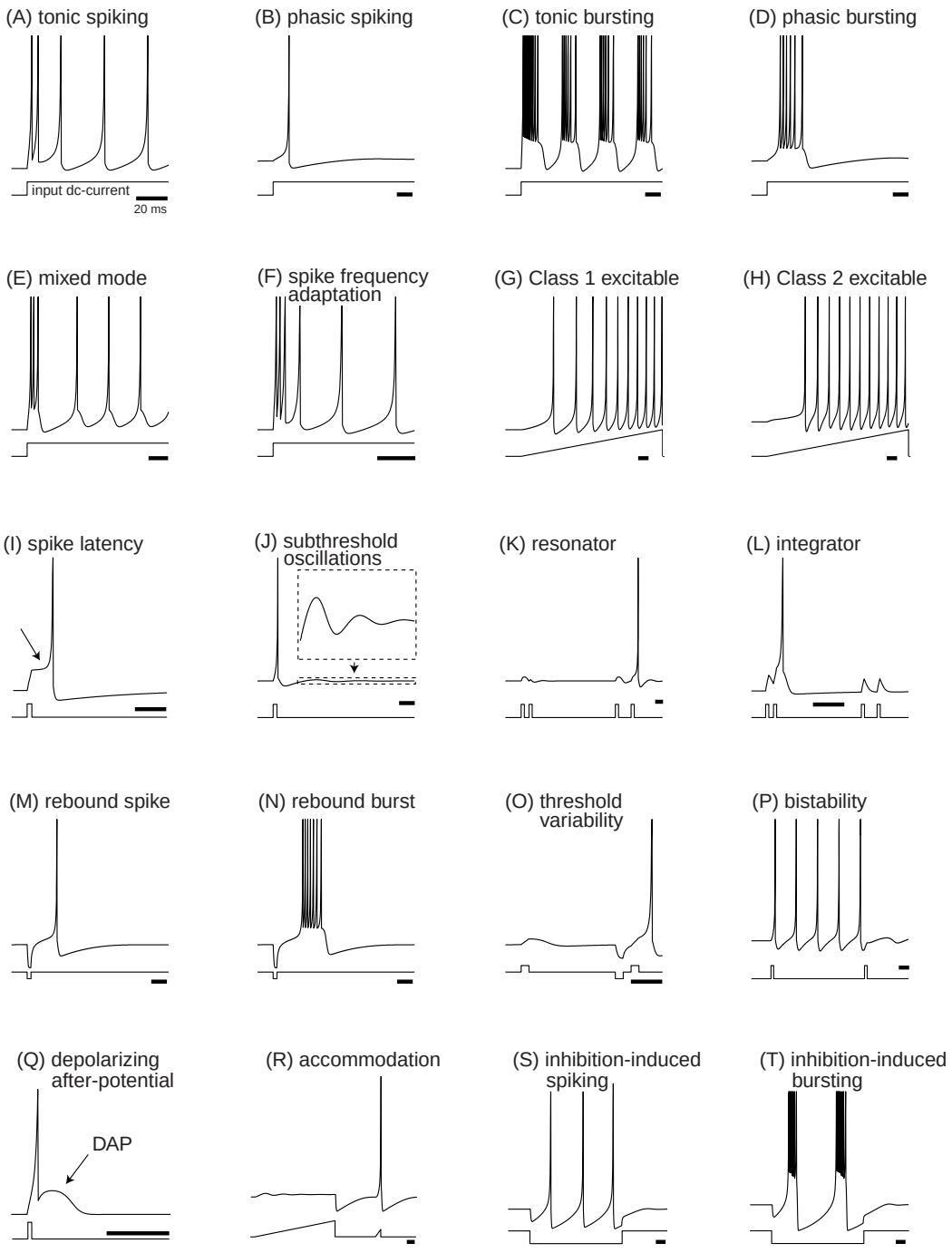

Figure 2.6: Examples of all neuro-computational features discussed in this section. The input current is plotted beneath the neuronal response and the horizontal bar denotes 20*ms*. No model may exhibit all these 20 neurocomputational properties simultaneously as some of them are mutually exclusive. All the neuronal responses showed here were obtained using a simple spiking model with only four easily tunable parameters [Izh03]. Electronic version of this figure and reproduction permissions are freely available at www.izhikevich.com.

n. **Rebound Burst:** The neuron fires post-inhibitory bursts. It is believed that such bursts contribute to the sleep oscillations in the thalamo-cortical system.

o. **Threshold Variability:** A biological neuron has a variable threshold that depends on the prior activity of the neurons. Therefore, it may not fire in response to a single brief excitatory pulse, but will fire if a brief inhibitory input is applied just before the same excitatory pulse.

p. **Bi-stability of Resting and Spiking States:** The neuron exhibits two stable modes of operation: resting and tonic spiking (or even bursting). The modes are switched by an excitatory or inhibitory pulse.

q. **Depolarizing After-Potentials:** After firing a spike, the membrane potential of a neuron may exhibit a prolonged after-hyperpolarization (AHP) or a prolonged depolarized after-potential (DAP). In any case, such a neuron has shortened refractory period and becomes superexcitable.

r. **Accommodation:** The neuron looses its sensitivity to a strong but slowly increasing stimulus (flat ramp) and does not fire, altought it may fire in response to a weak but fast increasing stimulus (step ramp).

s. **Inhibition-Induced Spiking:** The neuron remains quiescent as long as there is no input, but fires when hyperpolarized by an inhibitory input or an injected current.

t. **Inhibition-Induced Bursting:** Similar to the previous case, the neuron can fire tonic bursts of spikes in response to a steady hyperpolarization.

2.5 Conclusions

After this brief presentation we can evaluate what to simulate in order to have a spiking neuron model. It is also possible to have an idea about how complex a nervous system can be, even that of a simple animal. Now, once we know what characteristics to simulate, the next chapter will introduce a few mathematical neuronal models, which will permit us to simulate these characteristics with varying degrees of details, ranging from the simple *integrate-and-fire* model to the far more complex *compartmental model*.

Chapter 3

Spiking Neuron Models

3.1 Introduction

Since neural activity may be described at several levels of abstraction, the choice of the model depends on this level. On one hand, considering the microscopic level, there is a large number of ion channels, pores in the cell membrane which open and close depending on the voltage and the presence (or absence) of various chemical messenger molecules. Compartmental models, where each small segment of a neuron is described by a set of ionic equations, aim at a description of these processes. On the other hand, when choosing a higher level of abstraction, there is no need to worry about the spatial structure of a neuron or about the exact ionic mechanisms. We only have to consider the neuron as a simple homogeneous unit, which generates spikes if the total excitation is sufficiently large, as with rate models. It is also clear that the choice of which model to use is a very important decision that must be based on what kind of phenomena or neuronal behavior we wish to simulate.

In this rather superficial explanation, we divided the spiking neuron models into three main classes, namely threshold-fire, conductance based and compartmental models [GK02b]. Each one of these classes will be treated here, with emphasis on the first two, as the third is quite complex to fit in the scope of this work. A fourth class, namely rate model [Maa97, Ger99], was also considered, although it may not be classified as a genuine spiking model.

3.2 Threshold-Fire Models

The threshold-fire models are based on the temporal summation of all contributions to the membrane potential $u(t)$ received from all presynaptic neurons. If this contribution exceeds a threshold ϑ, then the postsynaptic neuron will fire. In the simplest case one assumes that the PSPs simply sum up linearly in the soma. In this section we will discuss two of these models, the integrate-and-fire and the spike response model – SRM [Ger01, GK02b].

3.2.1 Integrate-and-Fire Model – I&F

The integrate-and-fire neuron [MB98] is perhaps the most used and well-known example of a formal spiking neuron model. The basic model is also called leaky-integrate-and-fire (LIF) [GK02b] because the membrane is assumed to be leaky due to ion channels, such that after a PSP the membrane potential approaches again a reset potential u_{rest}. This model is also known as the *linear* integrate-and-fire neuron.

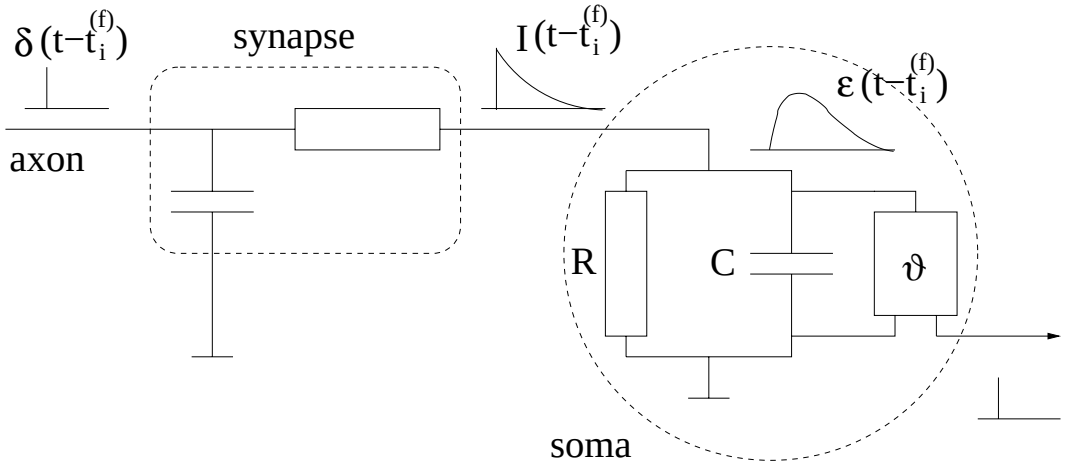

Figure 3.1: Integrate-and-fire neuron (from [MB98]).

The model is composed by the RC circuit shown inside the circle on the right-hand side of the diagram in figure 3.1. The circuit is charged by an input current I. If the voltage across the capacitor C reaches a threshold ϑ, the circuit is shunted and a pulse is transmitted to other neurons (lower right). A pulse sent out by a presynaptic neuron and travelling on the presynaptic axon (left), is low-pass filtered first (middle) before it is fed as a current pulse $I(t - t_i^{(f)})$ into the

integrate-and-fire circuit. The voltage response of the RC circuit to the presynaptic pulse is the postsynaptic potential $\epsilon(t - t_i^{(f)})$. The driving current splits into two components, one charging the capacitor, the other going through the resistor, yielding the equation (3.1).

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt} \tag{3.1}$$

Making the time constant $\tau_m = RC$ we can rewrite 3.1 in the form of equation (3.2),

$$\tau_m \frac{du}{dt} = -u(t) + RI(t) \tag{3.2}$$

where τ_m is the membrane time constant. It is easy to see that the voltage $u(t)$ across the capacitor C is the membrane potential.

Since equation (3.2) is a first-order linear differential equation and cannot describe full neuronal spiking behavior, it must be supplemented by a threshold condition. A pulse will be emitted at $t = t^{(f)}$ when the threshold ϑ is reached, i.e., when $u(t^{(f)}) = \vartheta$. The form of the spike is not described explicitly, only the firing time $t^{(f)}$ is used. Immediately after $t^{(f)}$, the potential is reset to u_{rest} and the integration starts over again from this point [MB98, GK02b]. The reset potential is defined by equation (3.3).

$$\lim_{t \to t^{(f)};\, t > t^{(f)}} u(t) = u_{rest} \tag{3.3}$$

The model behavior is then ruled by equations (3.2) and (3.3).

Considering now an integrate-and-fire neuron with a constant input current I_0 and a rest potential $u_{rest} = 0$ and integrating (3.2) with the initial condition $u(t^{(1)}) = u_{rest} = 0$, we obtain equation (3.4).

$$u(t) = RI_0 \left[1 - e^{\left(-\frac{t - t^{(1)}}{\tau_m}\right)}\right] \tag{3.4}$$

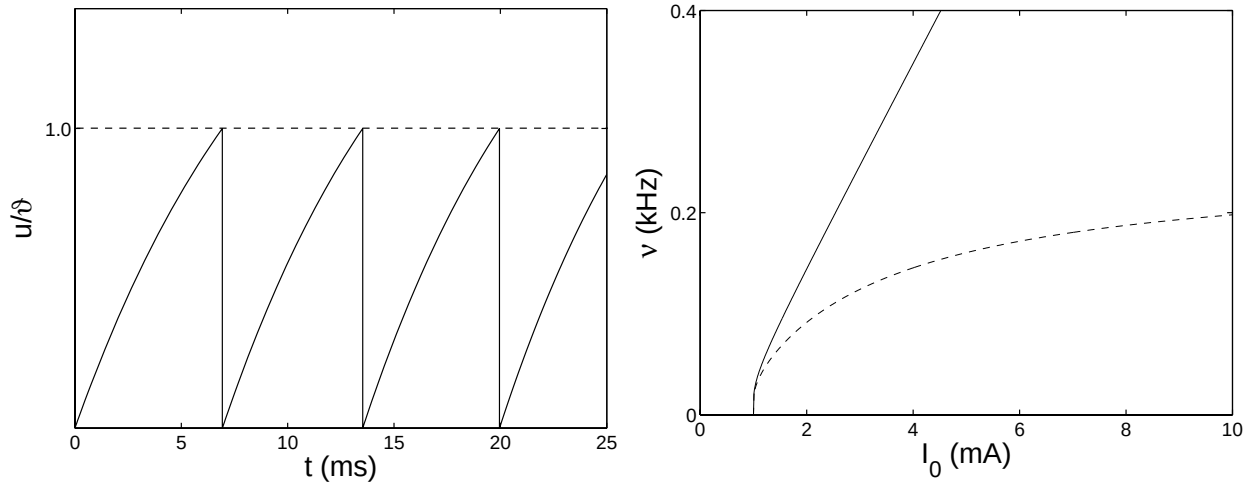

Figure 3.2: **Left:** Example of the membrane potential of an integrate-and-fire neuron excited by a constant input current $I_0 = 2mA$, with $\tau = 10ms$, $u_{rest} = 0$ and $R = 1\omega$. **Right:** Variation of the firing rate as a function of the exciting current in an integrate-and-fire neuron, with the same parameters. In the solid line curve no absolute refractoriness was used and in the dashed line curve a $4ms$ absolute refractoriness was used.

Assuming that a spike occurred at $t = t^{(1)}$, no further spike will occur as long as $u(t) < \vartheta$. For the production of a spike at $t = t^{(2)}$ we must have the condition $u(t^{(2)}) = \vartheta$, or

$$\vartheta = RI_0 \left[1 - e^{\left(-\frac{t^{(2)} - t^{(2)}}{\tau_m}\right)}\right] \tag{3.5}$$

Making $T = t^{(2)} - t^{(1)}$ and solving (3.5)

$$T = \tau_m \ln \frac{RI_0}{RI_0 - \vartheta} \tag{3.6}$$

We can see that the neuron will keep firing with period T as long as the constant current I_0 is applied. To take into account an absolute refractoriness Δ^{abs}, we just have to add it to the period T. Since in this case the frequency is the mean firing rate ν, we have

$$\nu = \left[\Delta^{abs} + \tau_m \ln \frac{RI_0}{RI_0 - \vartheta}\right]^{-1} \tag{3.7}$$

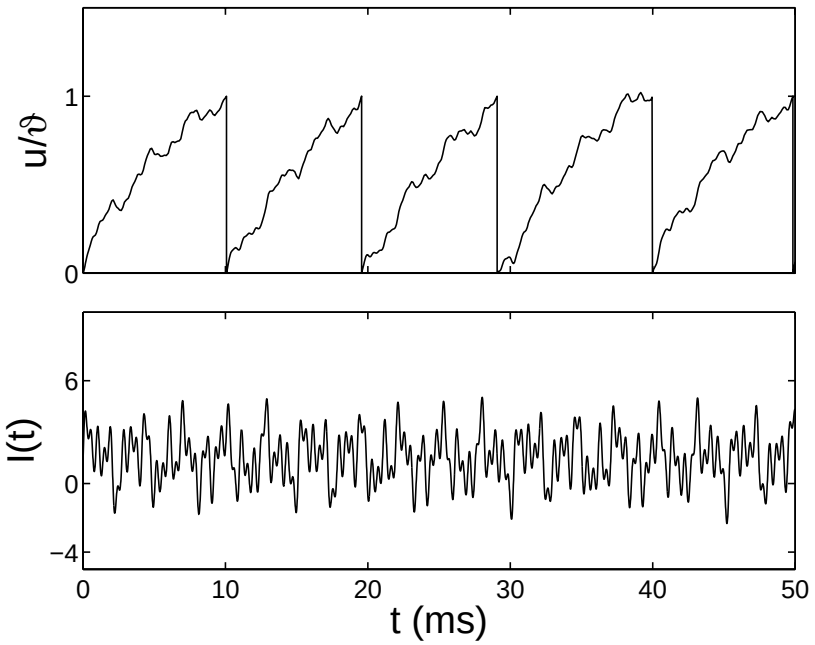

Figure 3.3: Example of the membrane potential of a linear integrate-and-fire neuron excited by a variable input current, with $\tau = 10ms$, $u_{rest} = 0$ and $R = 1\omega$. As suggested in [GK02b], the input current is the sum of four sinusoidal components with randomly chosen frequencies and a positive current $I_0 = 1.5mA$.

A variation of the linear integrate-and-fire model includes a second linear equation, describing the dynamics of a high-threshold K^+ ionic current, enabling the spike frequency adaptation. Other variations are the integrate-and-fire-or-burst neuron [Izh04, Coo03, Rul02], used to model cortical neurons, and the resonate-and-fire model [Izh01].

A more general form of equation (3.1), known as the *nonlinear* integrate-and-fire neuron, is given by equation (3.2.1),

$$\tau \frac{du}{dt} = F(u) + G(u)I \tag{3.8}$$

where $F(u)$ is a voltage dependent decay constant and $G(u)$ is a voltage dependent input resistance. A variation of this model is the *quadratic* integrate-and-fire model [GK02b, Izh04], described by equation (3.2.1).

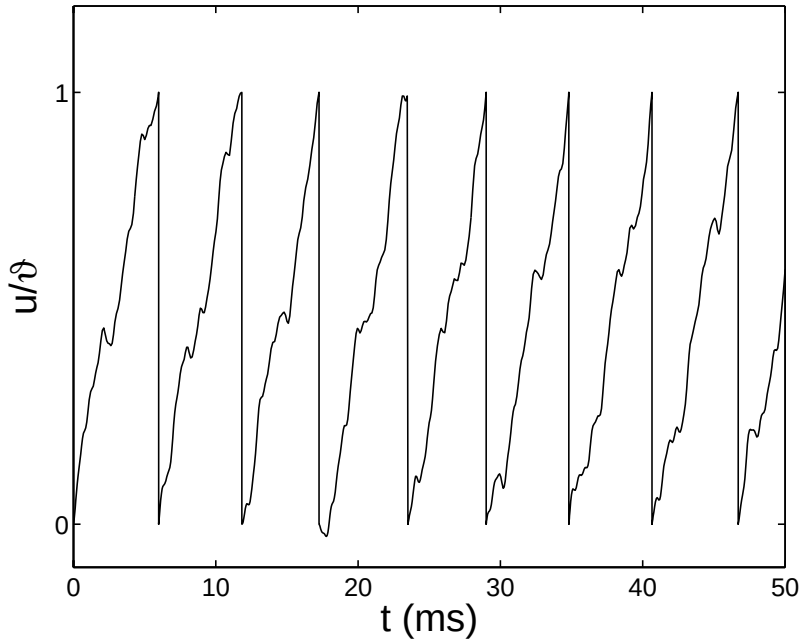

Figure 3.4: Example of the membrane potential of a quadratic integrate-and-fire neuron excited by the same input showed in figure 3.3. Here $u_{rest} = 0$ and $u_c = 0.5$.

$$\tau \frac{du}{dt} = a_0(u - u_{rest})(u - u_c) + RI \tag{3.9}$$

where $a_0 > 0$ is a constant and $u_c > u_{rest}$ is a critical voltage, above which $\frac{du}{dt} > 0$. This model is also known as Θ-neuron or the Ermentrout-Kopell canonical model (when written in its trigonometrical form) [Izh04]. This model has spike latencies, activity-dependent threshold and bi-stability of resting and tonic spiking modes [Izh04].

3.2.2 Spike Response Model – SRM

The state of a neuron j is described by a single state variable $u_j(t)$. If $u_j(t)$ crosses from below ($\frac{du}{dt} > 0$) a threshold ϑ at a moment $t_j^{(f)}$, then a spike is generated. As already stated in (2.1), the set of all firing times of neuron j is denoted by

$$F_j = \{t_j^{(f)}; 1 \leq f \leq n\} = \{t | u_j(t) = \vartheta\} \tag{3.10}$$

We can see in figure 2.5 that there are two different processes influencing the value of the state variable $u_j(t)$. The first is the contribution of all presynaptic neurons $i \in \Gamma_j$, which models the neuron response to presynaptic spikes. It is given by equation (3.11).

$$h(t) = \sum_{i \in \Gamma_j} \sum_{t_i^{(f)} \in \Gamma_j} w_{ij}\, \epsilon_{ij}\, (t - \hat{t}_j\,,\; t - t_i^{(f)} - \Delta_{ij}^{ax}) + I_j^{ext} \tag{3.11}$$

where w_{ij} is the synaptic efficacy, I_j^{ext} is an external excitation, and ϵ_{ij} is the PSP, as shown in figure 2.4 and equations (2.2) and (2.3). The postsynaptic potential can either be excitatory (EPSP) or inhibitory (IPSP). In most cases, Δ^{ax} is neglected and does not appear in the equations. To avoid confusion, we will use $s = t - t_i^{(f)} - \Delta_{ij}^{ax}$.

The second process is the reset of $u_j(t)$ till it reaches u_{rest}, including the negative overshoot which typically follows a spike (the refractory period). This term describes the response of neuron j to its own spikes and is modeled by a response kernel $\eta_j(t - \hat{t}_j)$, which is a function only of the last firing time of j ($\hat{t}_j$). More formally, all previous firing times of j should be considered, as in $\sum \eta_j(t - t_j^{(f)})$, but the approach adopted above does not alter the results [MB98].

Now, if we take into account the linear response of the membrane potential to an external driving current I^{ext}, the expression for the last term of (3.11) is then given by equation (3.12).

$$\int_0^\infty \kappa(t - \hat{t}_j\,, s) I^{ext}(t - s) ds \tag{3.12}$$

The total postsynaptic potential is also

$$h(t|\hat{t}_j) = \sum_i w_{ij} \sum_{t_i^{(f)}} \epsilon_{ij}(t - \hat{t}_j\,, s) + \int_0^\infty \kappa(t - \hat{t}_j\,, s) I^{ext}(t - s) ds \tag{3.13}$$

This dependence of the elapsed time since the last spike at $\hat{t}_j$ can be explained by the fact that immediately after a spike, many ion channels are open so that the membrane resistance is reduced [GK02b].

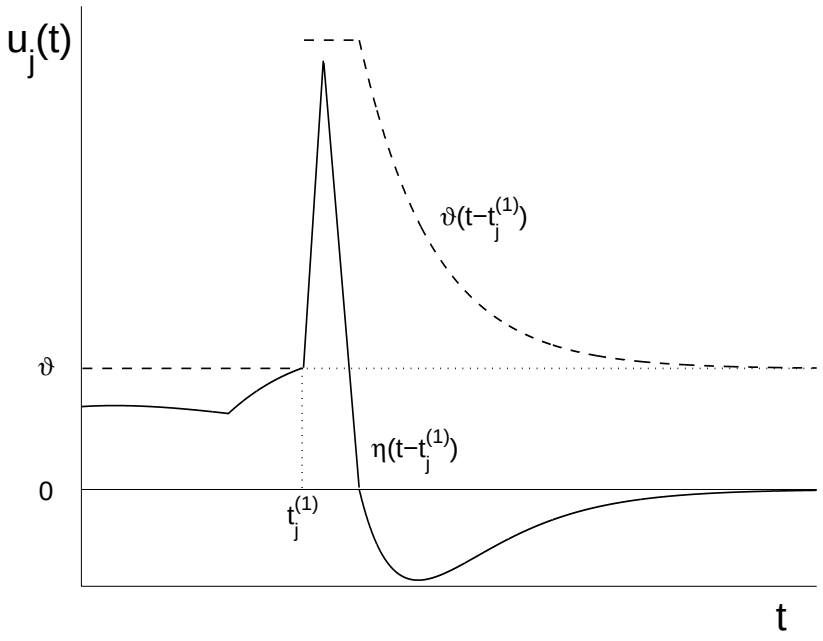

Figure 3.5: This is the same as figure 2.5, but the threshold was altered to show the effect of the dynamic threshold.

The state $u_j(t)$ of model neuron j at time t is given by the linear superposition of all contributions and can be written as in equation (3.14).

$$u_j(t) = \eta(t - \hat{t}_j) + h(t|\hat{t}_j) \tag{3.14}$$

Besides the use of the η kernel, refractoriness can be also modeled by an increase in the threshold after a spike so that $\vartheta \longrightarrow \vartheta(t - \hat{t}_j)$. If the objective is preventing the neuron from firing, an increase in the threshold is equivalent to the hyperpolarization of the membrane potential, as can be seen in figure 3.5. This is called *dynamic threshold* [MB98, GK02b].

A simplified Spike Response Model, SRM_0 for short [MB98], can be constructed with little modifications to the complete model, as shown in figure 3.6. First we suppose that ϵ_{ij} is the same for all presynaptic neurons and independent of the first argument $t - \hat{t}_j$. Then we make the κ kernel also independent of the first argument, so we can rewrite (3.14) as equation (3.15).

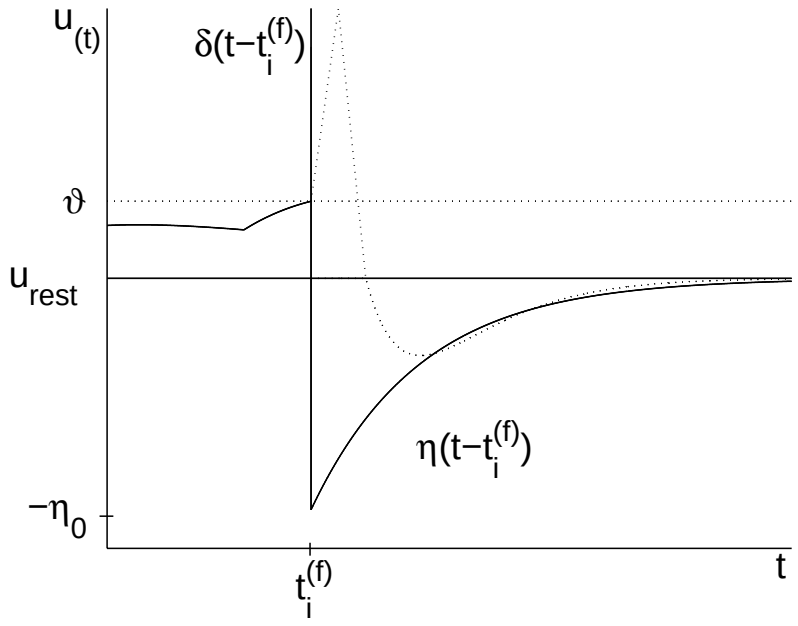

Figure 3.6: The SRM_0 model, where the spike was replaced by a δ-pulse and $\Delta^{ax} = 0$ (from [GK02b]).

$$u_j(t) = \eta(t - \hat{t}_j) + \sum_i w_{ij} \sum_{t_i^{(f)}} \epsilon_0(s) + \int_0^\infty \kappa_0(s) I^{ext}(t - s) ds \tag{3.15}$$

It can be demonstrated that the Spike Response Model, with the appropriate kernels, is equivalent to the integrate-and-fire model. Furthermore, with a different choice of kernels, the Spike Response Model also approximates the Hodgkin-Huxley equations with time-dependent input [MB98, GK02b].

3.3 Conductance-Based Models

This is a quite complex class of neuron models, which are usually based on the simulation of the intricate behavior of ionic channels. As these channels open and close, their conductances change accordingly, yielding a set of differential equations describing the process. The variations among the models in this class is mostly due to the choice of which channels to use and the parameters of the resulting differential equations. This section will be focused on the classic model of Hodgkin

and Huxley [HH52] and some of its variations.

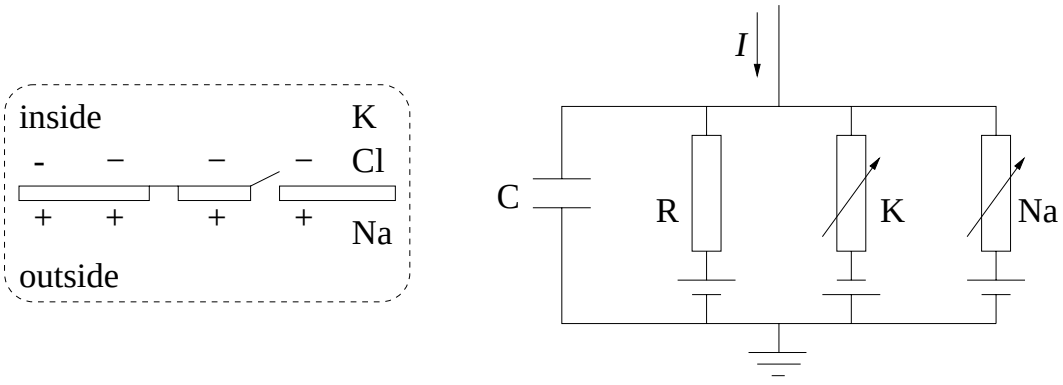

Figure 3.7: A schematic diagram for the Hodgkin-Huxley model (from [MB98]).

3.3.1 Hodgkin-Huxley Model

The Hodgkin-Huxley model is generally accepted as the classic description of a conductance model. Hodgkin and Huxley [HH52], based on their experiments with the giant axon of the squid, found three types of ionic current, namely sodium (Na), potassium (K) and a leak current. The first two types are controlled by specific voltage dependent ion channels. The third type takes care of other channels and consists mainly of Cl^- ions.

Taking figure 3.7 as a reference, the model can be described as a RC-circuit. The cell membrane is a good insulator and acts as a capacitor. Besides the capacitor, there are three resistances, one for each ion channel considered in the model. The potassium and sodium channels are voltage dependent. As explained before in section 2.3, the different ion concentrations inside and outside the cell yields the *Nernst Potential*, represented by the batteries. As the total current must be equal to the sum of the currents of the branches, we have

$$I(t) = I_c(t) + \sum_{ch} I_{ch}(t) \tag{3.16}$$

Since $I_c = C\frac{du}{dt}$

$$C\frac{du}{dt} = -\sum_{ch} I_{ch} + I(t) \tag{3.17}$$

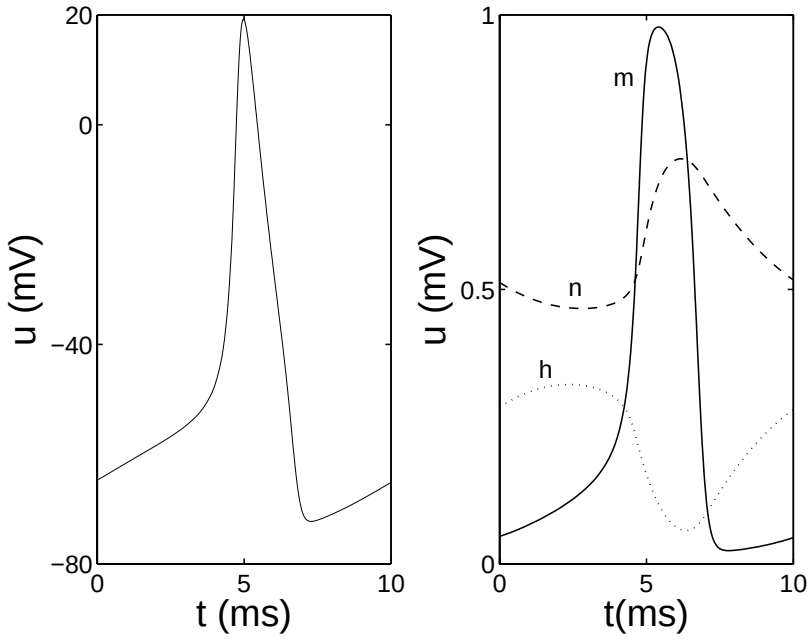

Figure 3.8: **Left:** Action potential generated by a Hodgkin-Huxley model with the parameters shown in the table 3.3.1. **Right:** Values of the parameters *m*, *n*, and *h* for the same interval.

The membrane potential is given by $u(t)$ and $\sum_{ch} I_{ch}$ is the sum of the ionic currents passing through the cell membrane.

Table 3.1: Parameters of the Hodgkin-Huxley equation. The usual units used for these parameters and variables are mS/cm^2 for conductances, mV for the equilibrium potentials, and μF for capacitances.

x	$E_x(mV)$	$g_x(mS/cm^2)$
Na	50	120
K	-77	36
L	-54.4	0.3

Characterizing each channel by its conductance (or resistance), the leak conductance (g_L) is constant and the other two (g_K and g_{Na}) are time and voltage dependent. Furthermore, considering that a channel is either open or closed, three more variables were added [HH52], yielding equation (3.18) for the sum of the three currents.

Table 3.2: Parameters of the equations for α_x and β_x.

x	$\alpha_x(u/mV)$	$\beta_x(u/mV)$
m	$(2.5-0.1u)/[\exp(2.5-0.1\mathrm{u})-1]$	$4\exp(-u/18)$
h	$0.07\exp(-u/20)$	$[1/[\exp(3-0.1u)+1]$
n	$(0.1-0.01u)/[\exp(1-0.1u)-1]$	$0.125\exp(-u/80)$

$$\sum_{ch} I_{ch} = g_{Na}m^3h(u-E_{Na}) + g_K n^4(u-E_K) + g_L(u-E_L) \tag{3.18}$$

If the channels were always open, the currents could be defined simply by $I_x = g_x u$ [Koc99]. Equation (3.17) can then be written as

$$C\frac{du}{dt} = -[g_{Na}m^3h(u-E_{Na}) + g_K n^4(u-E_K) + g_L(u-E_L)] + I(t) \tag{3.19}$$

The three new variables are given by the following differential equations

$$\begin{aligned} \frac{dm}{dt} &= \alpha_m(u)(1-m) - \beta_m(u)m \\ \frac{dn}{dt} &= \alpha_n(u)(1-n) - \beta_n(u)n \\ \frac{dh}{dt} &= \alpha_h(u)(1-h) - \beta_h(u)h \end{aligned} \tag{3.20}$$

The functions α and β, shown in table 3.3.1, are empirical equations originated from the adaptations made to fit the experimental data. Equations (3.19) and (3.20) define the Hodgkin-Huxley model. Figure 3.9 shows the equilibrium functions and the time constants for the variables m, n, and h. The values for m_0, n_0, h_0, τ_m, τ_n, and τ_h are given by equations (3.21).

$$
\begin{aligned}
x_0 &= \frac{\alpha_x(u)}{[\alpha_x(u)+\beta_x(u)]} \\
\tau_x &= \frac{1}{[\alpha_x(u)+\beta_x(u)]}
\end{aligned}
\tag{3.21}
$$

where x stands for m, n or h. Equation (3.20) is usually also written as

$$
\frac{dx}{dt} = \frac{x_0 - x}{\tau_x} \tag{3.22}
$$

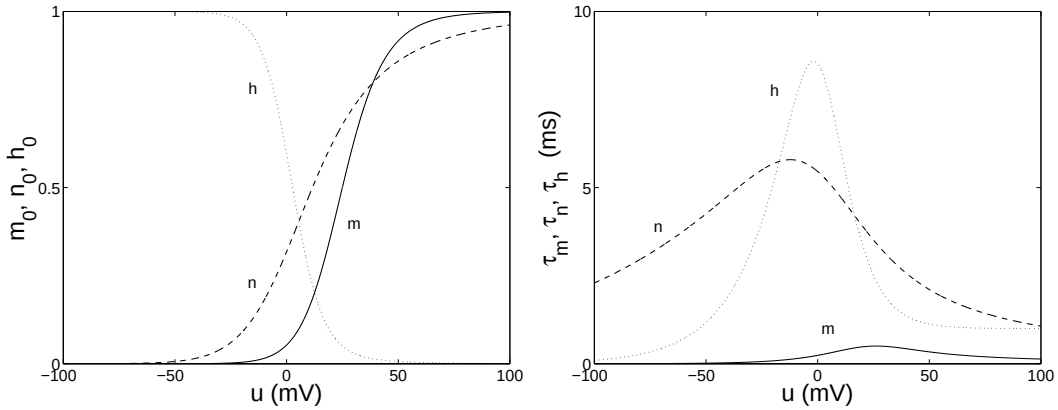

Figure 3.9: **Left:** Steady-state activation (m for sodium and n for potassium) and inactivation (h for sodium). **Right:** time constants as a function of the membrane potential. (from [Koc99, GK02b]).

With the previous equations and the right set of parameters, Hodgkin and Huxley were able to describe the enormous amount of data collected from experiments on the giant axon of the squid [Has00]. After this successful first step, there have subsequently been several modifications to the original equations in order to describe other experimental situations, giving origin to other models. These models differ mainly in the additional ion channels that have to be included (or excluded), specially those that account for Ca^{2+} and the slow components of the potassium current. As a general rule, for each type of ion channel i, a current $I_i = g_i x_i^{ni}(u - E_i)$ is added [MB98, Wil99, Koc99]. Here x is a variable with dynamics, like m, n, or h; g_i is the conductance parameter; n_i is an appropriate exponent; and E_i is the correspondent equilibrium potential.

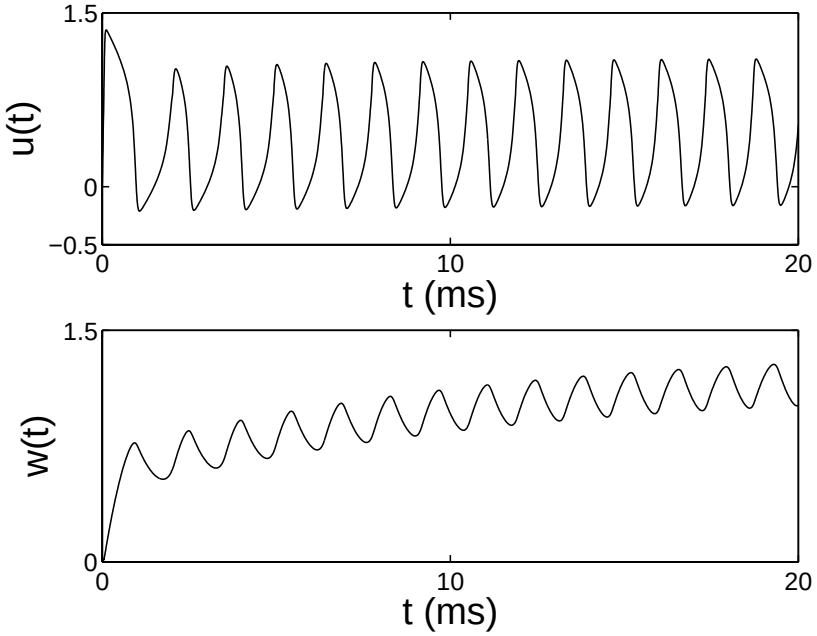

Figure 3.10: Example of the FitzHugh-Nagumo model with parameters $\alpha = 0.3$, $\epsilon = 0.03$, $\gamma = 0.4$, with an excitation current of 0.5μA and a threshold of 0.8mV.

3.3.2 FitzHugh-Nagumo Model

The FitzHugh-Nagumo model [Fit61, NAY62], described by equations (3.23), is a simplification of the Hodgkin-Huxley model where the set of four equations is reduced to a system of two equations, i.e., a cubic equation representing the fast variable (u) and a linear equation for the slow variable (w).

$$\begin{aligned} \epsilon \frac{du}{dt} &= -u(u-\alpha)(u-1) - w + I \\ \frac{dw}{dt} &= u - \gamma w \end{aligned} \tag{3.23}$$

The advantage of the reduction of the number of equations to a two-dimensional set of equations is that it allows a systematic phase plane analysis of the model. The two-dimensional set of equations is also called a Bonhoeffer / Van der Pol oscillator [Wil99].

3.3.3 Morris-Lecar Model

This 2D model was suggested by Morris and Lecar [ML81] to describe oscillations in a barnacle giant muscle fiber. The model is given by equations (3.24), where one expression is for the membrane potential u and the other for the potassium gating variable n:

$$\begin{aligned} C\frac{du}{dt} &= I - g_{Ca}m_{\infty}(u)(u - E_{Ca}) - g_K n(u - E_k) - g_L(u - E_L) \\ \frac{dn}{dt} &= \phi\frac{[n_{\infty}(u) - n]}{\tau_n(u)} \end{aligned} \tag{3.24}$$

where the steady states for Ca^{2+} and K^+ and the time constant τ_n are:

$$m_{\infty} = \frac{1}{1 + exp[-(u - u1)/u2]} \tag{3.25}$$

$$n_{\infty} = \frac{1}{1 + exp[-(u - u3)/u4]} \tag{3.26}$$

$$\tau_n(u) = \frac{1}{\cosh\left(\frac{u - u2}{2u4}\right)} \tag{3.27}$$

Sometimes τ_n is referred to as $[\lambda_n(u)]^{-1}$. Except for a change in the parameters, m_{∞} and n_{∞} are functionally identical.

3.3.4 Hindmarsh-Rose Model

The general form of the Hindmarsh-Rose model of thalamic neuron [HR82, HR84] is given by equations (3.28).

$$\begin{aligned} \frac{du}{dt} &= v - F(u) + I - w \\ \frac{dv}{dt} &= G(u) - v \\ \frac{dw}{dt} &= \frac{[H(u) - w]}{\tau} \end{aligned} \tag{3.28}$$

where F, G and H are specific functions that must be correctly chosen to produce the desired results. Depending on their choice, the model can exhibit all the properties shown in figure 2.6 [Izh04]. For instance, equations (3.29) were used in [BC84].

$$\begin{aligned} \frac{du}{dt} &= v - au^3 + bu^2 + I - w \\ \frac{dv}{dt} &= c - du^2 - v \\ \frac{dw}{dt} &= \frac{[s(u - u_0) - w]}{\tau} \end{aligned} \tag{3.29}$$

where $F(u) = au^3 - bu^2$, $G(u) = c - du^2$, and $H(u) = s(u - u_0)$.

3.3.5 Izhikevit Model

This model, proposed in [Izh03], can exhibit all the properties shown in figure 2.6. The model uses equations (3.30).

$$\begin{aligned} \frac{du}{dt} &= 0.04u^2 + 5u + 140 - w + I \\ \frac{dw}{dt} &= a(bu - w) \end{aligned} \tag{3.30}$$

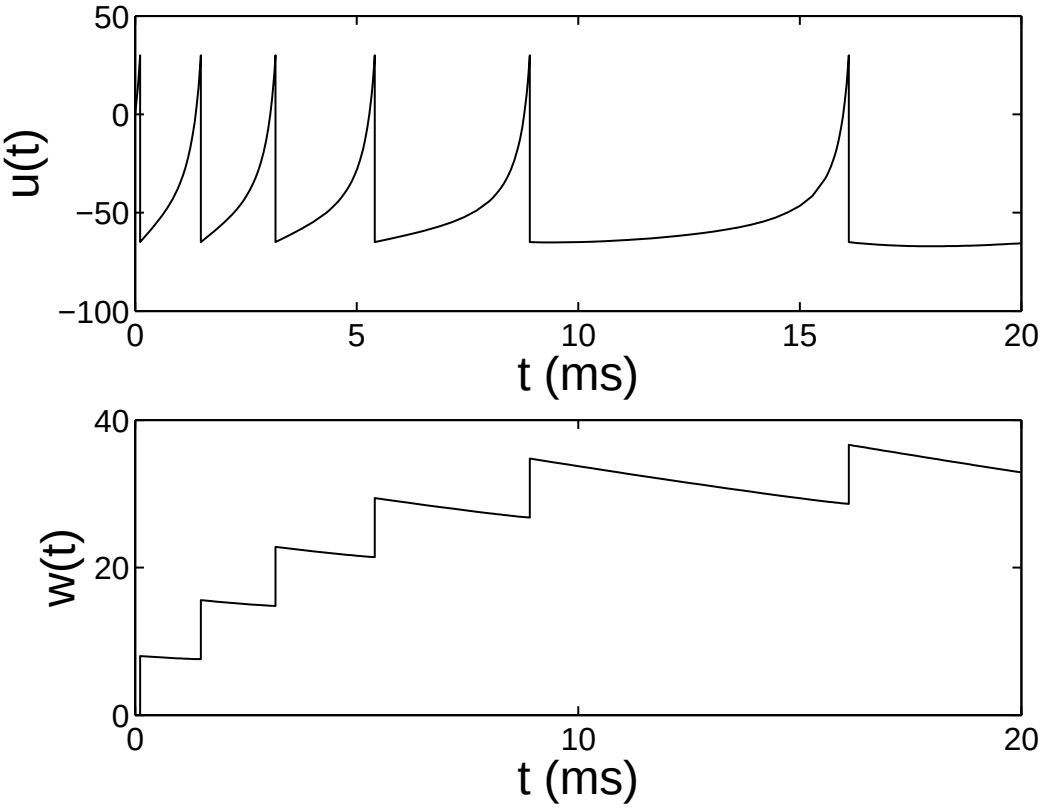

Figure 3.11: Example of the Izhikevit model with parameters $a = 0.02$, $b = 0.2$, $c = -65$, and $d = 8$, with an excitation current of 50μA and a threshold of 30mV.

with the resetting after-spike

$$\text{if } u \geq +30\,\text{mV}, \quad \text{then } \begin{cases} u \leftarrow c \\ w \leftarrow w + d \end{cases} \tag{3.31}$$

The variable u represents the membrane potential and w is a membrane recovery variable, accounting for the activation of K^+ ionic currents and inactivation of Na^+ ionic currents. The model can exhibit firing patterns of all known types of cortical neurons with the choice of parameters a, b, c, and d given in [Izh03].

3.4 Compartmental Models

Since Ramón y Cajal [yC03] first described a neuron, the known complexity of this biological structure has been steadily increasing. This complexity is better captured by the compartmental models, which take into account the spatial structure of the dendritic tree and also model the synaptic transmission at a greater level of detail. With this approach, it is possible to consider other ion currents, beyond the Na^+ and K^+ currents incorporated in the Hodgkin-Huxley model (Ca^{2+} and Ca-mediated K currents which are related to neuronal adaptation [MB98]). Entire books have been written on this subject and it is not our aim to present a complete description of these neuron models, but just to introduce its basic concepts.

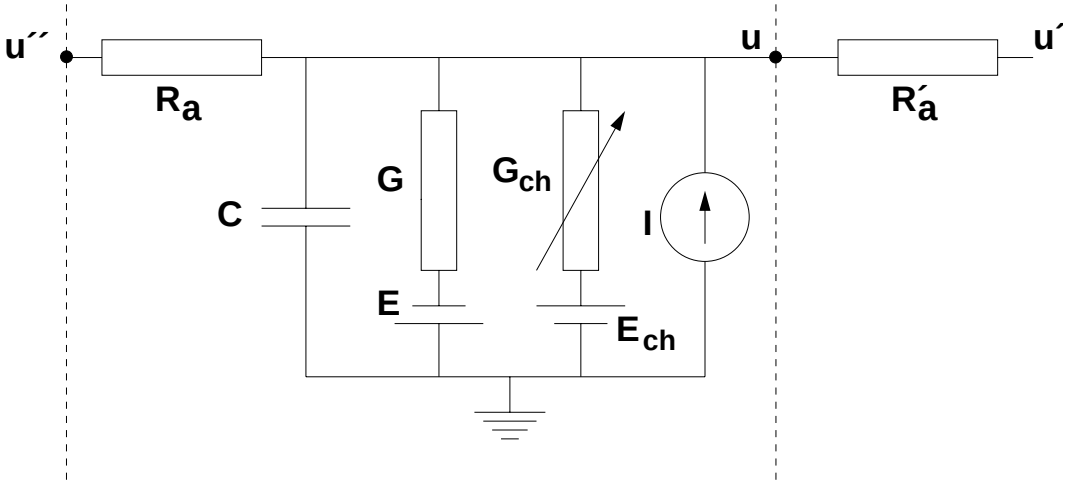

Figure 3.12: A "generic" equivalent circuit of a neural compartment (from [BB03]).

As already mentioned previously, the dendrites receive synaptic inputs from other neurons. These inputs activate ion channels in the dendrites, which create postsynaptic potentials that are propagated to the cell body (soma), where voltage-activated ion channels may create action potentials. Usually these channels are located in a region called *axon hillock*, at the base of the soma [KSJ00, PAF+04]. The axon then propagates the action potentials to its terminal branches, where they form synapses connecting to other neurons.

The basic idea of compartmental models is to divide these components into smaller uniform

components or *compartments*. Each compartment is then modeled with equations describing the equivalent electrical circuit. The use of appropriate differential equations for each compartment enables the simulation of their behavior as well as their interactions with other compartments. This notion of an equivalent electrical circuit for a small piece of cellular membrane is the basis for all compartmental models [BB03].

An equivalent circuit is shown in figure 3.12, where u is the membrane potential, which appears across the membrane capacitance C. It may cause a current flow through the axial resistance R_a (or R_a') into or out the compartment when there is a difference $u - u$" (or $u - u'$) between the two compartments. The resistor G_{ch} crossed by an arrow represents a generic variable channel conductance, specific to a particular ion combination. The corresponding equilibrium potential is represented by the battery in series with the conductance. The other resistor is the *leakage* conductance G, and the associated equilibrium potential E is typically close to u_{rest}. The current I represents an external input. The resulting differential equation is (3.32).

$$C\frac{du}{dt} = (E-u)G + \sum_{ch}(E_{ch}-u)G_{ch} + \frac{(u'-u)}{R_a'} + \frac{(u"-u)}{R_a} + I \tag{3.32}$$

Equation (3.32) describes a linear compartmental model. Other more complex compartmental models would also contain non-linear conductances for various ionic currents, or could model synaptic currents at a greater level of detail.

3.5 Rate Models

The rate models, also known as *sigmoidal units*, are the most traditional and widely used models for the analysis of learning and memory in artificial neural networks.

As already mentioned, the first choice to be made before the construction of a neuronal model is the level of abstraction and, therefore, the complexity of the model. The class of rate models, first proposed in [MP43], represents the highest abstraction level, as they neglect the pulse structure of the neuronal systems and the neural activity is described only in terms of spike rate. A good example of rate model is the Perceptron [Ros58, MP88].

The response ν_j of a neuron j is also a rate and depends on a sigmoidal function. The input to this function is the summation of all the excitations applied to j, i.e., all the output rates of the presynaptic neurons $i \in \Gamma_j$:

$$\nu_j = g\left(\sum_{i\in\Gamma_j} w_{ij}\nu_i\right) \tag{3.33}$$

where w_{ij} is the synaptic strength between neurons i and j. Equation (3.33) is the starting point of standard neural network theory [Hay99, BLdC00].

The dependence of a sigmoidal function may be explained by the equation (3.34), known as Naka-Rushton function [NR66], which relates the stimulus intensity p, the net postsynaptic potential, to the response or spike rate of a neuron observed in several visual areas (lateral geniculate, striate cortex, middle temporal cortex).

$$\nu(p) = \begin{cases} \dfrac{Mp^n}{\sigma^n + p^n} & \text{for } p \geq 0 \\ 0 & \text{for } p < 0 \end{cases} \tag{3.34}$$

In equation (3.34), ν is the spike rate response to the stimulus p; M is the maximum spike rate for very intense stimuli; *sigma* determines the point where $\nu(p)$ reaches half its maximum (semi-saturation point); and n determines the maximum slope of the function. A plot of this function is shown in figure 3.13, for $n = 2$, 4 and 6, $M = 100$, and $\sigma = 50$.

3.6 Conclusions

As one can grasp from the previous material, a real neuron is a very complex entity and its mathematical modelling may become a quite challenging task, depending on how deep into the details we intend to through. This depth is determined by the behaviors and features the modeled neurons must have. The models mentioned here are but a few among several other models used for general or special purposes, e.g., [IMI99, CBC02, RH05, SKA01, SS02, TNS02]

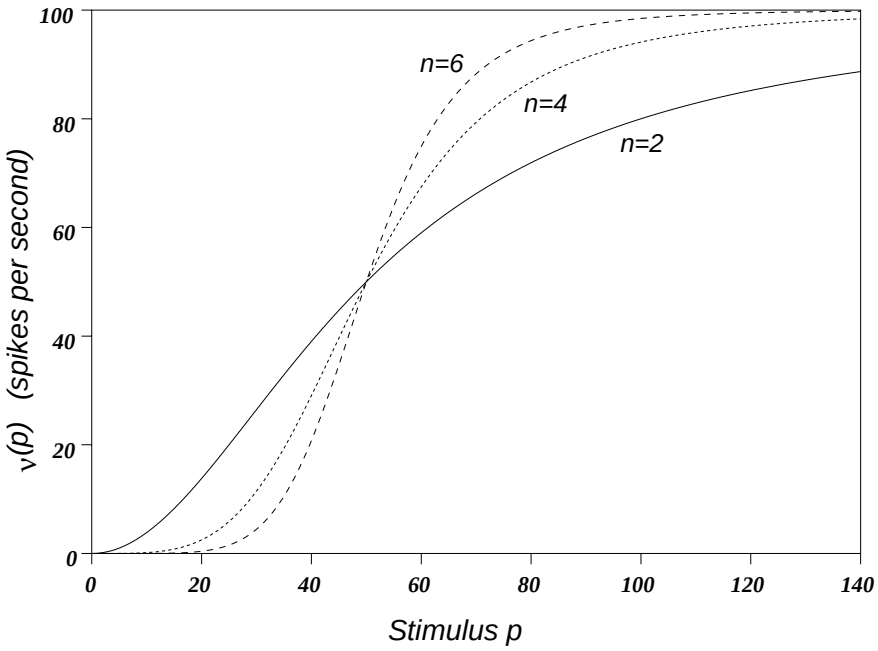

Figure 3.13: A plot of equation (3.34) for $M = 100$, $\sigma = 50$, and $n = 2$, 4 and 6.

In the following chapters the models presented here will be explored and some kind of neural dynamics will be studied.

Chapter 4

Neurodynamics

4.1 Introduction

In the previous chapters we reviewed the basic components of the nervous systems, giving a glimpse of all their underlying complexity. Following, we then discussed some of the mathematical models that permit us to simulate these systems, from the more simple to the complex ones. The objective of this chapter is to briefly discuss those aspects tied to the practical application of the neuron models to simulate the dynamical behavior of real neural systems [And04].

The complete dynamical behavior of a neural system is usually described by high-dimensional nonlinear differential equations, whose results are difficult to visualize, and even more difficult to analyze [Kin91]. Fortunately, there are some models that are a reduction of the more complex models to two dimensions, like the Morris-Lecar [ML81] or the FitzHugh-Nagumo [Fit61, NAY62] models. The use of these simplified equations makes it easier (or less difficult) to study and simulate the nonlinear phenomena of neural systems, like bifurcation and hysteresis.

The nature of the stability and steady states of a nonlinear system can be determined by the eigenvalues of the associated Jacobian matrix, evaluated at the equilibrium points [ASY00]. Given a nonlinear system described by the equation

$$\frac{d\mathbf{X}}{dt} = \mathbf{F}(\mathbf{X})$$

then the equilibrium point $\mathbf{X_{eq}}$ is a solution to $\mathbf{F}(\mathbf{X_{eq}}) = \mathbf{0}$ [ASY00].

One of the simplest equations that have been proposed for spike generation are the FitzHug-Nagumo equations (3.23), presented here with esplicit time constants as equations (4.1). It is worth of noting that the time constant for u is usually more than ten times times faster than that for w, reflecting the fact that the process in the axon is faster than the recovery process [Wil99].

$$\begin{aligned} \frac{du}{dt} &= \frac{1}{\tau_u}\left(u - \frac{u^3}{K_1} - w + I\right) \\ \frac{dw}{dt} &= \frac{1}{\tau_w}(-w + K_2 u + K_3) \end{aligned} \tag{4.1}$$

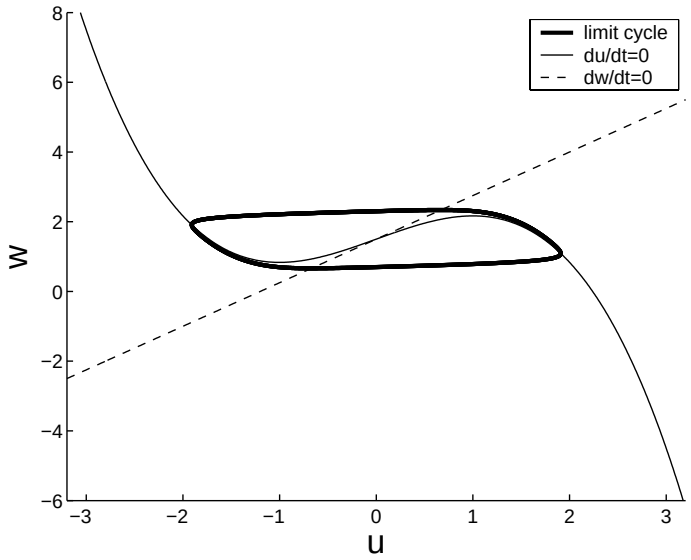

Figure 4.1: Phase plane showing the limit cycle (bold line) and nullclines for $\frac{du}{dt} = 0$ (solid line) and $\frac{dw}{dt} = 0$ (solid line) using the FitzHugh-Nagumo equations, with a external stimulus of $I = 1.5$ and parameters $\tau_u = 0.09$, $\tau_w = 1.20$, $K_1 = 3$, $K_2 = 1.25$ and $K_3 = 1.50$.

To give a geometric picture of the dynamics of a model, the use of phase plane (or phase space) is quite helpfull. A phase plane is the plot of pairs of points $(u(t), w(t))$ for a range of t values, as shown in figure 4.1 for the FitzHug-Nagumo equations, with an external input $I = 1.5$. In this figure we can see the nullclines, drawn for the values of u and w for which the right sides of equations (4.1) are zero, and an equilibrium point, where the nullclines cross each other. In this

case there is only one equilibrium point. Also shown in figure 4.1 is a limit cycle, which is a closed orbit in the phase plane, indicating periodic behavior [DA01].

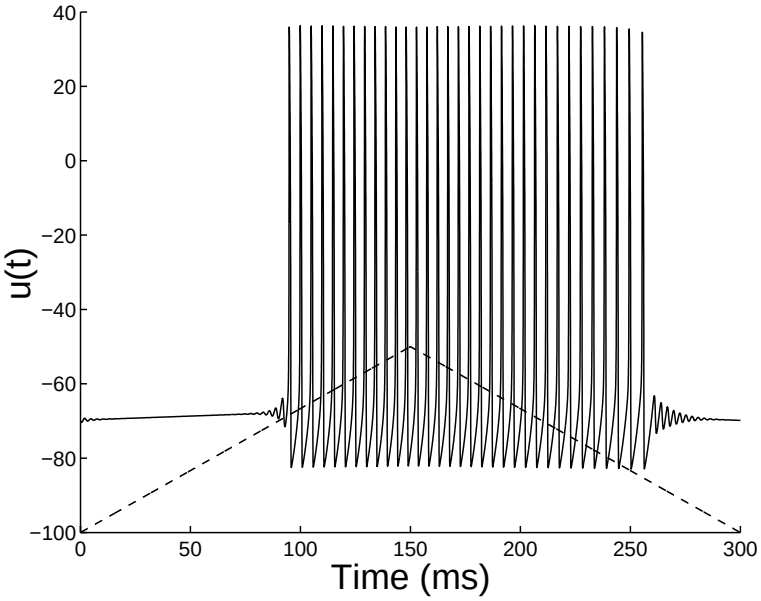

Figure 4.2: Hysteresis in the squid axon produced in response of a triangular current ramp, with values ranging from $I = 0$ to $I = 0.25nA$ (showed here out of scale, only for reference). The parameters used in equations (4.2) were $C = 0.8\mu F/cm^2$ and $\tau = 1.9$

4.2 Example 1: Hysteresis

The hysteresis is an important nonlinear phenomenon, often seen in the presence of a subcritical Hopf bifurcation [ASY00, HI01]. Concerning neural systems, its existence was practically demonstrate in a squid axon [GLR80] by stimulating it with a current that began at 0 and increased linearly up to a maximum value.

$$\begin{aligned} C\frac{du}{dt} &= -(17.81 + 47.7u + 32.63u^2)(u - 0.55) - 26.0w(u + 0.92) + I \\ \frac{dw}{dt} &= \frac{1}{\tau}(-w + 1.35u + 1.03) \end{aligned} \tag{4.2}$$

To emphasize the importance of neural modeling, we repeat this experiment using equations (4.2) [Wil99]. An illustration of the results is shown in figure 4.2. Hysteresis occur because the present state of a nonlinear system is determined not only by the present state of the stimulation but also by the history of stimulation.

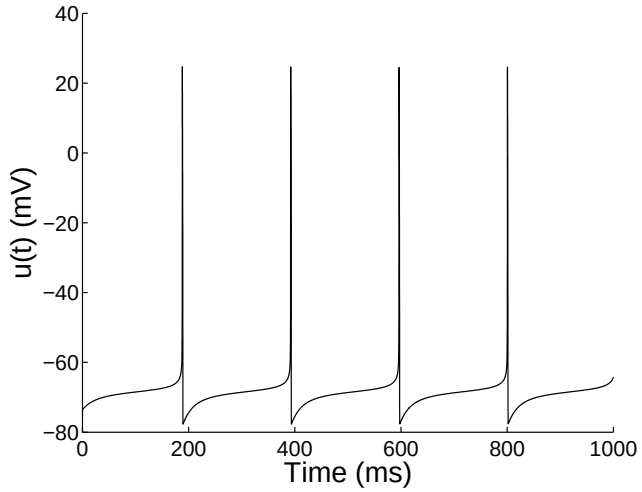

Figure 4.3: Spike train with a rate of 4.9 spikes per second, showing a class 1 exitable neuron

4.3 Example 2: Class 1 and Class 2 Excitabilities

One of the features of a neuron described in chapter 2 was class 1 and class 2 excitabilities. Almost all mammalians neurons are of the first type [KSJ00], while the squid axon is of the second class. Neurons of the class 1 may fire at arbitrary low rate and those of class 2 do not. The ability to fire at low rates is attributed to a rapid transient K^+ current [Wil99], which is found in a wide range of neurons, including mammalian and human neocortical neurons. Using equations (4.2) slightly modified to accommodate a quadratic equation for the recovery variable w, yielding equations (4.3), it is possible to simulate a slow firing neuron. These new equations permit the approximation of many effects of the transient current [RH89]. The results obtained with the parameters values used here approximate those exhibit by a human neocortical neuron [Wil99].

Figure 4.3 shows a spike train with a rate of 4.9 spikes per second produced by equations (4.3) with a capacitance $C = 1.0\mu F/cm^2$, an input current $I = 0.22nA$ and a time constant $\tau = 5.6ms$.

$$
\begin{aligned}
C\frac{du}{dt} &= -(17.81 + 47.58u + 33.8u^2)(u - 0.48) - 26.0w(u + 0.95) + I \\
\frac{dw}{dt} &= \frac{1}{\tau}(-w + 1.29u + 0.79 + 0.33(u + 0.38)^2)
\end{aligned}
\tag{4.3}
$$

Using again equations (4.3) with the same parameters values as above and applying an input current $I = 0$ and $I = 0.6nA$, it is possible to illustrate the occurrence of a saddle-node bifurcation. The results for both current values are plotted in figure 4.4, where we can see three steady states for $I = 0$ (when the nullclines cross each other), namely a stable node, a saddle point, and a unstable spiral point.

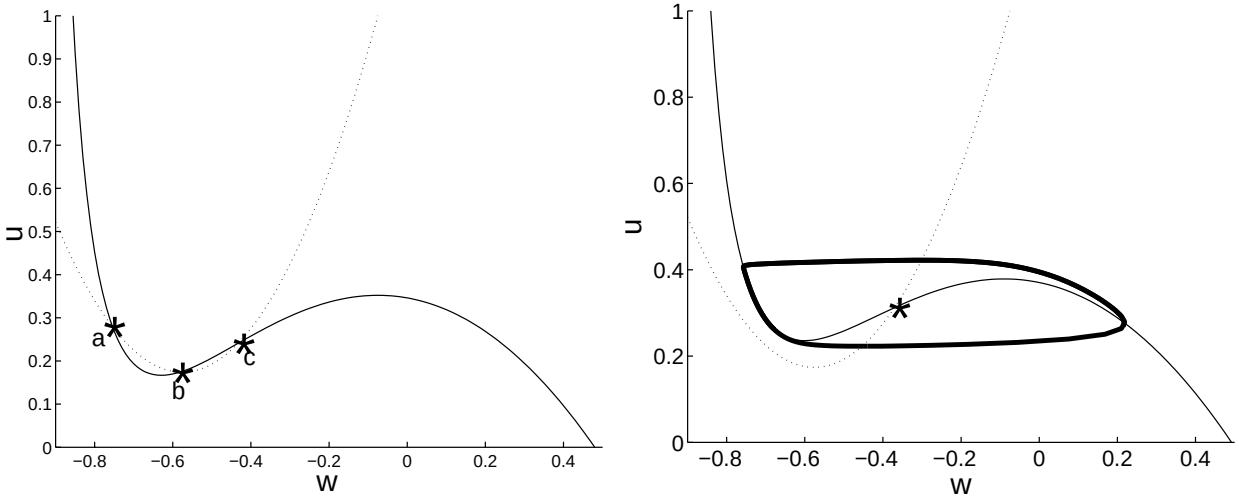

Figure 4.4: Phase plane showing the limit cycle and nullclines for the human neuron for $I = 0$ (left) and $I = 0.6nA$ (right).

.

As I is increased the behavior of the systems changes qualitatively and the nullclines are modified so that the asymptotically stable node and the saddle point move toward each other and, at a specific transition point, both coalesce and vanish and only the unstable spiral point remains [GK02b] (see figure 4.4). This transition point between stationary and oscillatory states is called bifurcation and, in this special case, a saddle-node bifurcation. Because of the dependence

of a bifurcation on the value of I, it is some times called the *bifurcation parameter*.

Comparing figures 4.1 and 4.4 it is easy to see that in the first case there is only one fixed point, while in the second there are three. In equations 4.1) the only fixed point changes from stable to unstable at a critical value of I, denoting the occurrence of a (subcritical) Hopf bifurcation [GK02b]. The difference between these two situations is that with only one fixed point the transition to the oscillatory state occurs with a nonzero frequency (class 2), and thus with a discontinuity in the gain function (spike rate versus I). In the second case, the transition is smooth (class 1) and the gain function is continuous [HI01] (see figure 4.5).

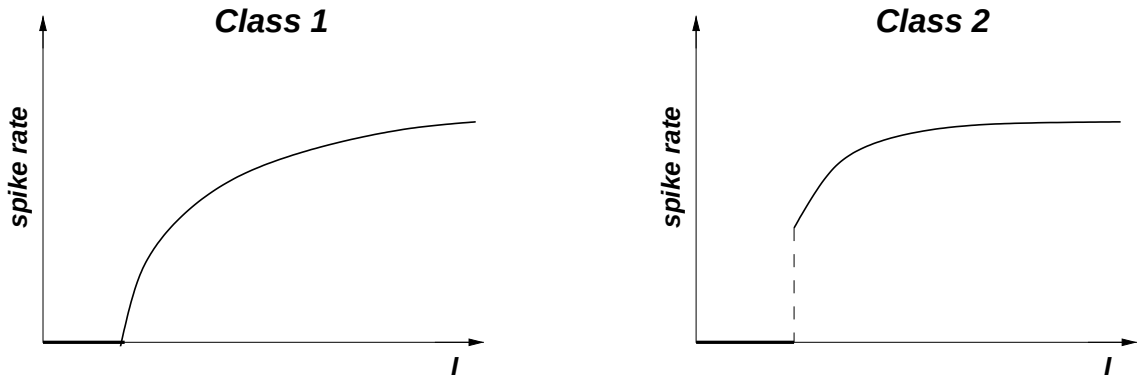

Figure 4.5: Gain function for Class 1 and 2 neurons. The class 1 curve is a continuous function, while the class 2 curve has a discontinuity.

4.4 Example 3: Neural Chaos

Applying a periodic stimulus $i(t) = I_0 + Asin(2\pi\omega t)$ instead of a constant current I, we add to equation (4.2) a sufficiently high dimensionality to permit chaos to occur. Indeed, with the same parameters values used to produce hysteresis, and $I_0 = 0.075nA$, $A = 0.007nA$ and $\omega = 264.6Hz$, a chaotic spike train will be generated, as shown in figure 4.6.

The presence of chaos in neural systems arise some important questions. For instance, we must consider that the sensitivity to initial conditions implies lack of long-term predictability, so that greater descriptive detail in a chaotic neural system will not necessarily lead to greater predictive capability. At the level of animal behavior this unpredictability can be of some advantage, as the prey fleeing from a predator may be using some adaptative application of neural chaos in

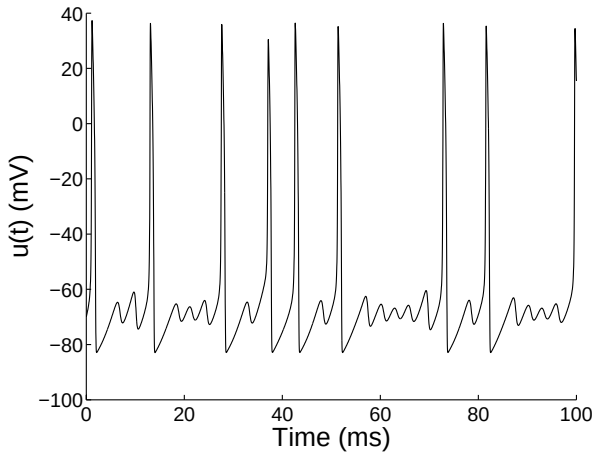

Figure 4.6: Chaotic spike train Phase plane showing the limit cycle and nullclines for the human neuron for $I = 0$ (left) and $I = 0.6nA$ (right).

.

motor control [Wil99]. In a more philosophic approach, since the cognitive functions are located in the cerebral cortex [KSJ00] and there may be chaotic neural subsystems, neural chaos may well explain unexpected human behavior. Indeed, there is already evidence of chaos in the monkey cortex and simulations have suggested that cortical networks with recurrent excitation or inhibition may exhibit chaos [vVS96].

4.5 Conclusions

From this brief discussion we can see that up to the present days, the usual application of the more detailed neuron models has been the simulation of real biological systems, which frequently display a nonlinear dynamical behavior. It is also worth noting that the elaboration and application of these more detailed models to do *engineering* tasks is still in its beginning, and there is a lot of work to be done to make the present models more tractable.

Chapter 5

Practical Applications Using SNN

5.1 Introduction

Once we have the knowledge of the basic principles and concepts of the real neurons and their various mathematical models, it is now time to apply this knowledge to build a SNN to carry on some practical tasks. Therefore, the main objective of this chapter is to show how neurons can encode and process information based only on spikes. Although there are several works dealing with clustering with SNN [SW99, Jin04, KIKY03], the examples given here use the unsupervised clustering method suggested in [Hop95] and implemented in [Ruf98] and [BPK02], with some minor modifications and variations. All implementations were written in MATLAB scripts [HL03], some of which are listed in Appendix A.

Before building a SNN, however, we have to explore three important issues, i.e., information encoding, learning method and network architecture, which will be discussed in the first sections. After that we will use the SNN to carry on some clustering tasks.

5.2 Information Encoding

The first question that arises when dealing with spiking neurons is how neurons encode information in their spike trains, since we are specially interested in a method to translate an analog value into spikes [HMR02], so we can process this information in a SNN. This fundamental issue

in neurophysiology remains still not completely solved [RWdRvSB97] and is extensively treated in several publications [SKdRvSB98, AT02, YS99, AA99, Koc99, PP95].

Although a line dividing the various coding schemes cannot always be clearly drawn [DA01], it is possible to distinguish essentially three different approaches [MB98, GK02b] in a very rough categorization:

- rate coding: the information is encoded in the firing rate of the neurons [Wil99].
- temporal coding: the information is encoded by the timing of the spikes. [Ruf98, Boh04]
- population coding: information is encoded by the activity of different pools (populations) of neurons, where a neuron may participate of several pools [Sni96, Ger99, GK02b].

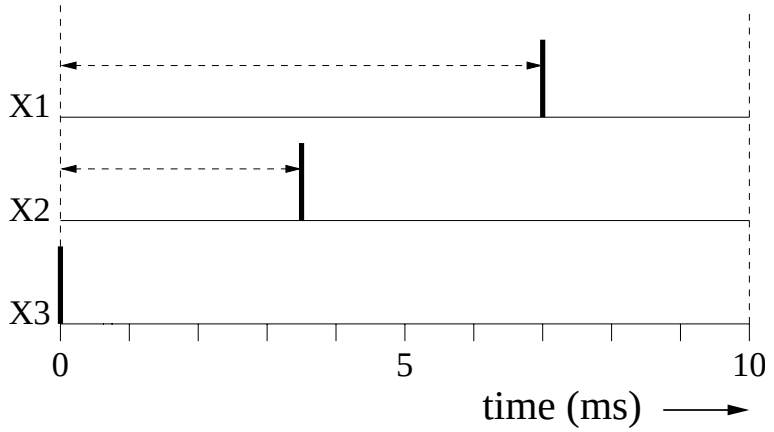

Figure 5.1: Simple temporal encoding scheme for two analog variables: $x_1 = 3.5$ and $x_2 = 7.0$, with x_3 as the reference, firing always at 0 in a coding interval of 10ms.

A very simple temporal coding method, suggested in [Hop95, Maa97, MB98], is to code an analog variable directly in a finite time interval. For example, we can code values varying from 0 to 20 simply by choosing an interval of 10ms and converting the analog values directly in proportional delays inside this interval, so that an analog value of 9.0 would correspond to a delay of 4.5ms. In this case, the analog value is encoded in the time interval between two or more spikes, and a neuron with a fixed firing time is needed to serve as a reference. Figure 5.1 shows the output of three spiking neurons used to encode two analog variables. Without the reference neuron, the

values 3.5 and 7.0 would be equal to the values 6.0 and 2.5, since both sets have the same inter-spike interval.

If we now fully connect these three input neurons to two output neurons, we will have a SNN like the one shown in figure 5.2, which is capable of correctly separating the two clusters shown in the right side of the figure. Although this is a too simple example, it is quite useful to demonstrate how real spiking neurons possibly work. The clustering here was made using only the axonal delays Δ_{ij}^{ax} (see Chapter 2) between the input and output neurons with all the weights equal to one.

In a SNN with m input and n output neurons, the center of the RBF-like output neuron J is given by the vector $\mathbf{c}_j = [c_{1,j}, c_{2,j}, \ldots, c_{m,j}]$, where $c_{i,j} = max\{\Delta_{ij}^{ax} | 1 \leq i \leq m\} - \Delta_{ij}^{ax}$. Similarly, the input vectors are defined as $\mathbf{x} = [x_1, x_2, \ldots, x_m]$, where $x_i = t_i - min\{t_i | 1 \leq i \leq m\}$ and t_i is the firing time of each input neuron [Ruf98]. The SNN used here has an EPSP (see figure 2.4) with a time constant $\tau = 1.84ms$ and a threshold $\vartheta = 2.04$. The lateral connection between the two output neurons is strongly inhibitory and will be explained in the following sections.

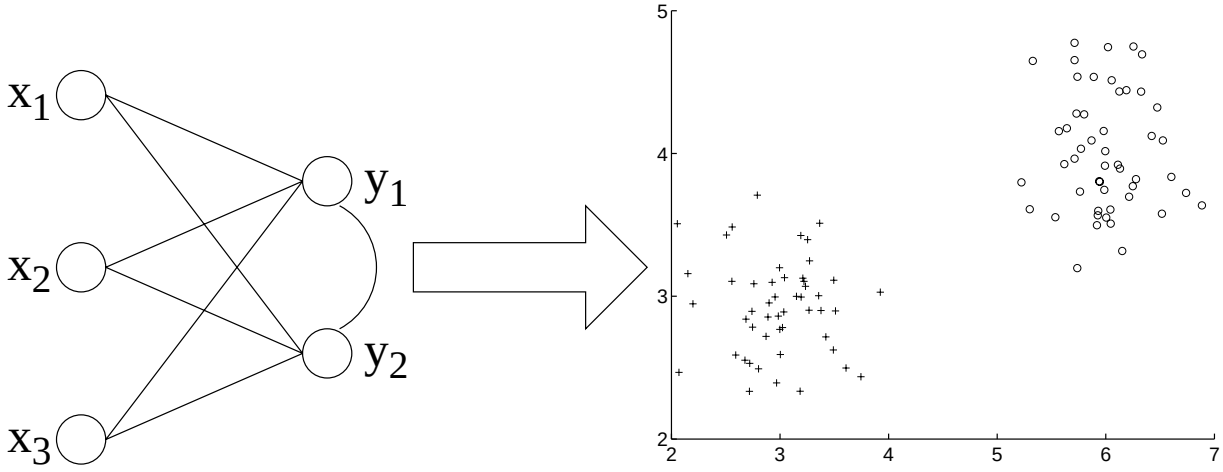

Figure 5.2: **Left**: SNN with a bi-dimensional input formed by three spiking neurons and two RBF-like output neurons. **Right**: two randomly generated clusters (crosses and circles), correctly separated by the SNN.

This encoding method can work well for a number of clusters less or equal to the number of dimensions, but its performance decreases when the number of clusters exceeds the number of

input dimensions. The proposed solution for this problem [BPK02] implemented here uses an encoding method based on population coding [Sni96, dKvdV01], which distributes an input variable over multiple input neurons. By this method, the input variables are encoded with graded and overlapping activation functions, modeled as local receptive fields. Figure 5.3 shows the encoding of the value 0.3. In this example, assuming that the time unit is millisecond, the value 0.3 was encoded with six neurons by delaying the firing of neurons 1 (5.564ms), 2 (1.287ms), 2 (0.250ms), 4 (3.783ms) and 5 (7.741ms). Neuron 6 does not fire at all, since the delay is above 9ms and lays in the no firing zone. It is easy to see that values close to 0.3 will cause neurons 2 and 3 to fire earlier than the others, meaning that the better a neuron is stimulated, the nearer to $t = 0ms$ it will fire. A value up to $t = 9ms$ is assigned to the less stimulated neurons, and above this limit the neuron does not fire at all (see figure 5.4).

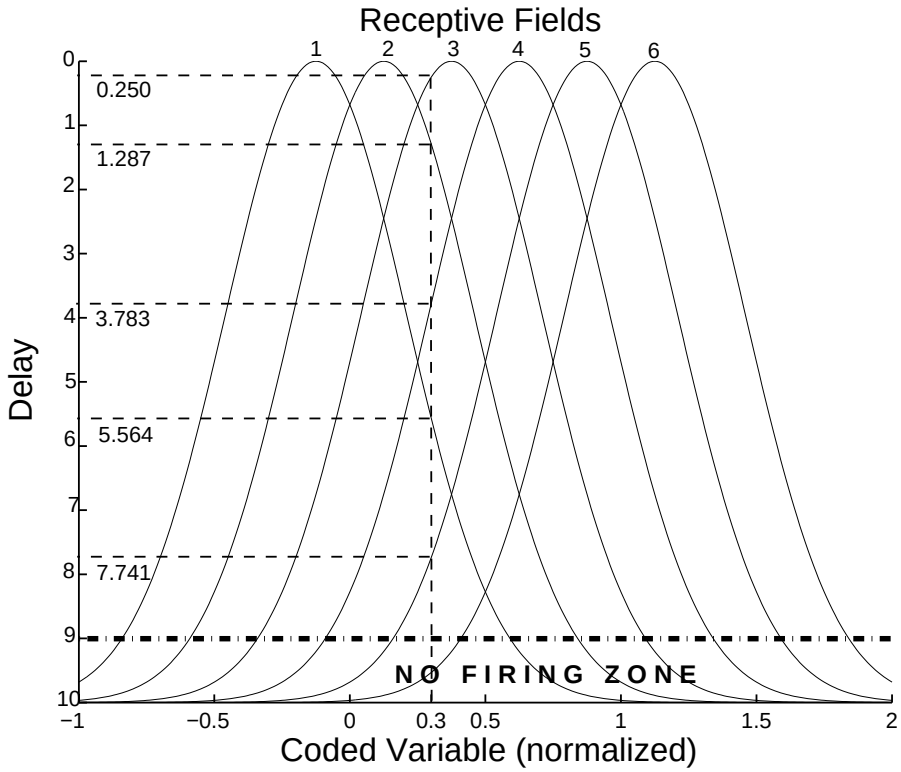

Figure 5.3: Continuous input variable encoded by means of local receptive fields.

Biologically, we can think of the input neurons as if they were some sort of sensory system sending signals proportional to their excitation, defined by the Gaussian receptive fields. These

neurons translate the sensory signals into delayed spikes and send them forward to the output neurons.

In this work, the encoding was made with the analog variables normalized in the interval [0,1] and the receptive fields equally distributed, with the centers of the first and the last receptive fields laying outside the coding interval [0,1], as shown in figure 5.3. There is another way to encode analog variables, very similar to the first, the only difference being that no center lays outside the coding interval and the width of the receptive fields is broader.

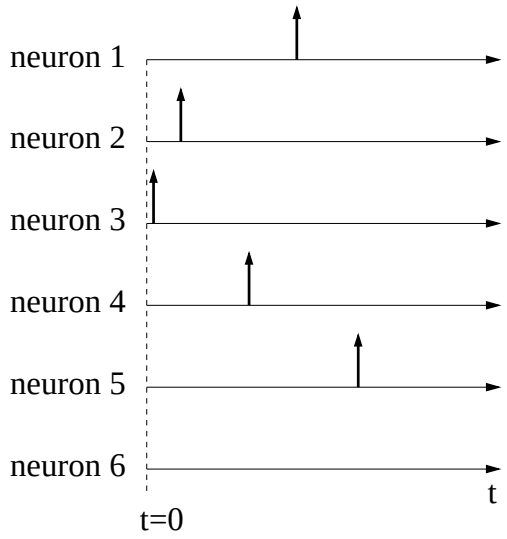

Figure 5.4: Spikes generated by the encoding scheme of the first type shown in figure 5.3.

Both types of encoding can be simultaneously used, like in figure 5.5, to enhance the range of detectable detail and provide multiscale sensitivity [BPK02]. The width σ and the centers c_i are defined by equations (5.1) and (5.2), for the first and second types, respectively. Unless otherwise mentioned, the value of γ used for the first type is 1.5 and 0.5 for the second.

$$\sigma = \frac{1}{\gamma(m+1)} \qquad c_i = \frac{i-1}{m-1} \tag{5.1}$$

$$\sigma = \frac{1}{\gamma(m-2)} \qquad c_i = \frac{i-1.5}{m-2} \tag{5.2}$$

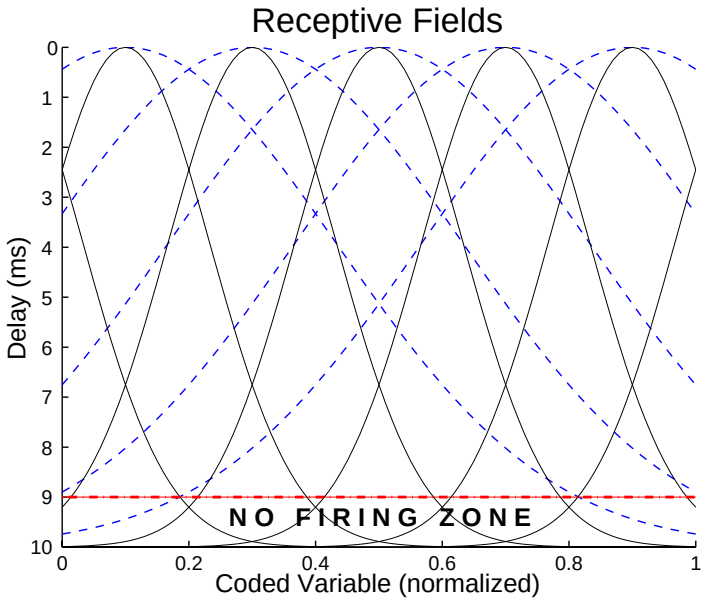

Figure 5.5: The two ways to encode continuous input variable by means of local receptive fields. The doted lines are wide receptive fields of the second type, with $\lambda = 0.5$

5.3 Learning

The approach presented here implements the Hebbian reinforcement learning method [Heb49] through a *winner-takes-all* algorithm [BLdC00, Hay99], whose practical application in SNN is discussed in [Ruf98, BPK02, dQ02] and a more theoretical approach is presented in [GK02a, BSS93]. In a clustering task, the learning process consists mainly of adapting the time delays, so that each output neuron represents an RBF center. This goal is achieved using a learning window [1] [KGvH99a, KGvH99b], which is defined as a function of the time interval Δt_{ij} between the firing times t_i and t_j. This function controls the learning process by updating the weights based on this time difference, as shown in equation (5.3), where Δw_{ij} is the amount by which the weights w_{ij} are increased or decreased and η is the learning rate.

$$\Delta w_{ij} = \eta L(\Delta t_{ij}) \tag{5.3}$$

[1] The learning window is also called learning function and both terms will be used here with the same meaning.

The learning function used here, shown in figure 5.6, is a Gaussian curve defined by the equation (5.4) [GK02a, KGvH99a, KGvH99b, LvH01]. It reinforces the synapse between neurons i and j if $\Delta t_{ij} < \nu$, and depresses the synapse if $\Delta t_{ij} > \nu$.

$$L(\Delta t) = (1+\beta)e^{\frac{(\Delta t-\alpha)^2}{2(\kappa-1)}} - \beta \tag{5.4}$$

where

$$\kappa = 1 - \frac{\nu^2}{2\ln\frac{\beta}{1+\beta}}$$

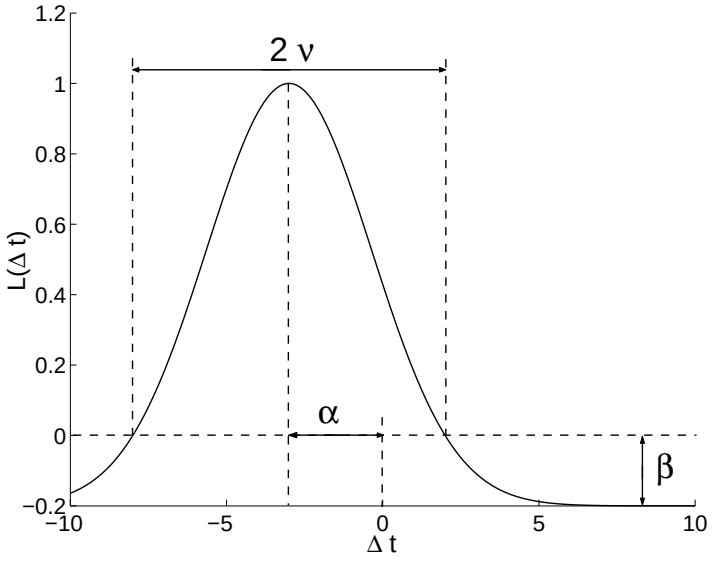

Figure 5.6: Gaussian learning function, with $\alpha = -3$, $\beta = 0.2$, and $\nu = 4$.

The learning window is defined by the following parameters:

- ν: this parameter, called here neighborhood, determines the width of the learning window where it crosses the zero line and affects the range of Δt_{ij}, inside which the weights are increased. Inside the neighborhood the weights are increased, otherwise they are decreased.

- β: this parameter determines the amount by which the weights will be reduced and corresponds to the part of the curve laying outside the neighborhood and bellow the zero line.

- α: because of the time constant τ of the EPSP (see figure 2.4), a neuron *i* firing exactly with *j* does not contribute to the firing of *j*, so the learning window must be shifted slightly to consider this time interval and to avoid reinforcing synapses that do not stimulate *j*.

5.4 Network Architecture

The architecture adopted here is almost the same as in a traditional neural network (e.g. a feed-forward network with the Perceptron model [Ros58]), with the neurons of the preceding layer fully connected to the neurons of the next layer. The first layer is composed by the receptive fields neurons and the second, by the output neurons, also called RBF neurons [BPK02]. If other layers are added, the neurons of the previous layer will act as the receptive fields of the next layer.

A first difference in relation to the traditional neural networks is that all output neurons are laterally connected to each other by means of a strongly inhibitory synapse. The purpose of this inhibitory synapse is to disable all other neurons after the first has fired, performing a *winner-takes-all* mechanism.

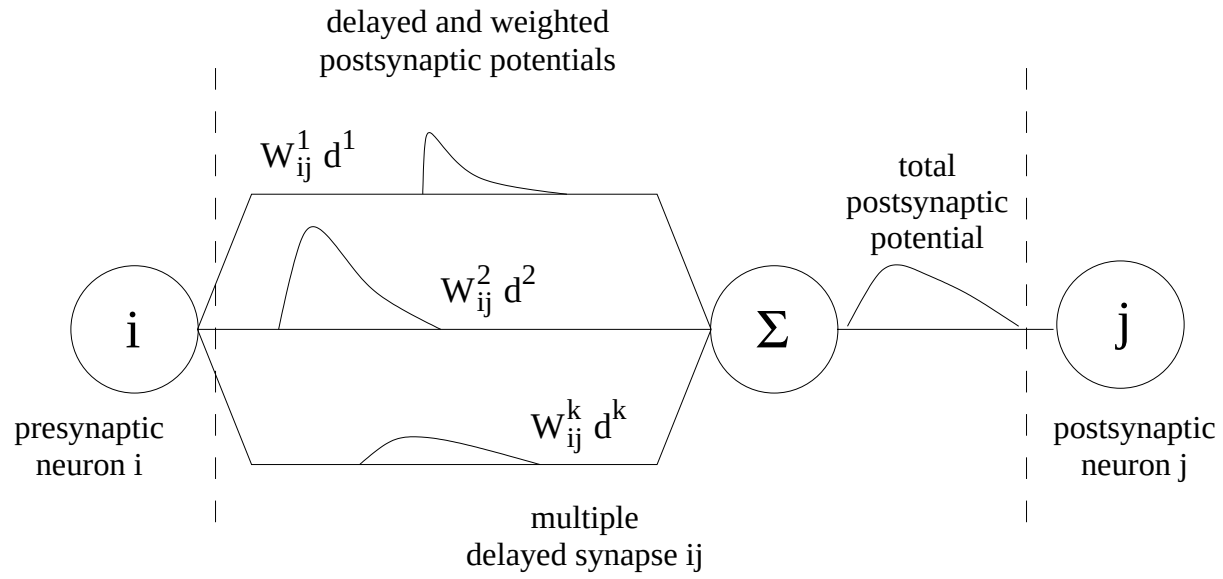

Figure 5.7: A single multiple synapse. The final EPSP is the summation of all potentials produced by the sub-synapses.

Another difference is the multiple synapse approach. The existence of multiple synapses is biologically plausible (e.g. [WZR98]), since in brain areas like the neocortex a single presynaptic axon makes several independent contacts with the postsynaptic neuron. The practical aspects of this theory has been recently discussed and neural computation using this approach has been already demonstrated [MZ99, NMZ01, NSD02, WF02].

Instead of a single synapse, with its specific delay and weight, this synapse model consists of many sub-synapses, each one with its own weight and delay, as shown in figure 5.7. Although the final effect is like the one of a single synapse (see figure 5.8), the use of multiple sinapses enables an adequate delay selection using the learning rule presented in the previous section. For each multiple synapse connecting neuron i to neuron j, with s sub-synapses, the resulting PSP is given by equation (5.5).

$$u_{ij}(t) = \sum_{k=1}^{s} w_{ij}^{k} \epsilon(t - t_i - d^k) \tag{5.5}$$

Where d^k has the same effect of the axonal delay represented by Δ_{ij}^{ax} in figure 2.4. The total contribution of all presynaptic neurons is then given by equation (5.6).

$$u_j(t) = \sum_{i \in \Gamma_j} \sum_{k=1}^{s} w_{ij}^{k} \epsilon(t - t_i - d^k) \tag{5.6}$$

The delays d^k are fixed for all sub-synapse k, varying from zero in 1ms fixed intervals. The most used arragements have 12 to 16 sub-synapses, with delays up to 15ms. These parameters are chosen experimentally, to produce the best results. The neuron model implemented is the SRM_0 [GK02b], with a strictly excitatory PSP modeled by equation (2.3), with $\tau = 3ms$. A leaky integrate-and-fire neuron could also be used with no modification in the network architecture.

It is worth mentioning that there is a fixed return delay [Ruf98] that should also to be include to take into account the effect of the feedback signaling from the membrane to the synapse. In our implementations we ignored this delay [BPK02]. Otherwise, we would have simply to add a new time delay to take into account this time interval.

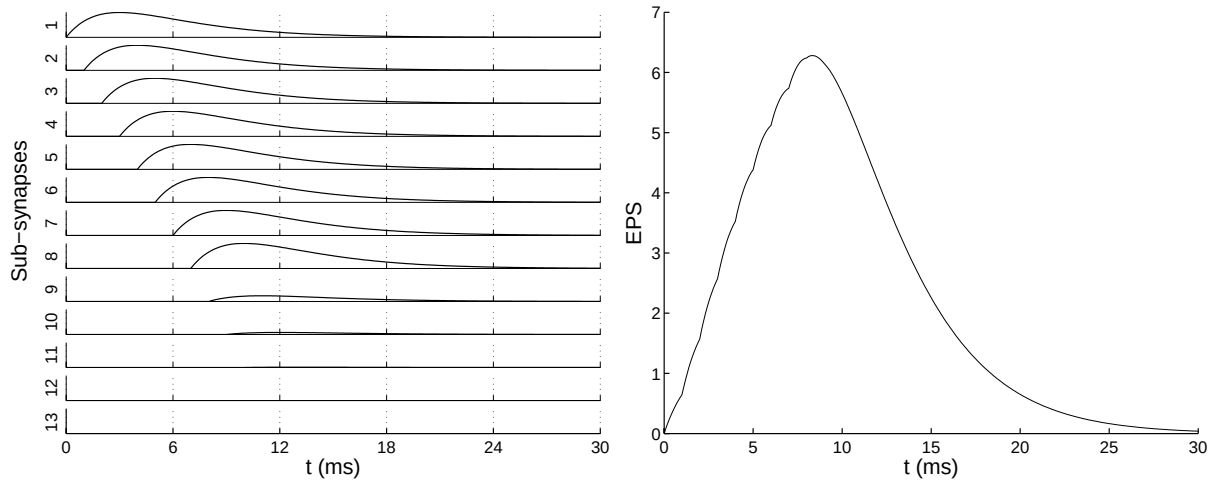

Figure 5.8: **Left**: The EPSP produced by each of the 13 sub-synapses. **Right**: The final EPSP is the summation of all potentials produced by the sub-synapses.

With this architecture, the resulting PSP is the temporal summation of the PSP produced by each sub-synapse. In figure 5.8 we show the EPSP produced by the sub-synapses connecting the input neuron 6 to the output neuron 1 in the SNN used to separate five clusters, discussed in the next section (see also input neuron 6 in the output neuron 1 graphic in figure 5.10)

5.5 Applying a SNN to a clustering Task

The first example developed here demonstrates the use of a SNN in a clustering task by correctly separating five bi-dimensional randomly generated clusters. The SNN uses the architecture explained in the previous section, with eight neurons encoding the inputs of each dimension, using a time interval of 10ms. Each cluster is represented by one of the five output neurons.

With this configuration we have $\Delta t_{ij} = t_i + d^k - t_j$ and the learning rule to be applied for updating all weights in all sub-synapses is given by equation (5.7).

$$\Delta w_{ij}^k = \eta\, L(t_i + d^k - t_j) \tag{5.7}$$

Since with population coding some of the neurons will not fire, these neurons must also be penalized, as well as the neurons firing in a time interval outside the neighborhood ν. The strategy

applied here was to adjust the weights of these neurons with $\Delta w_{ij}^{k} = -\eta$. It is also important to note that the values of the weights were limited to the interval [0,1], i.e., $0 \leq w_{ij}^{k} \leq 1$, to avoid instability of the SNN.

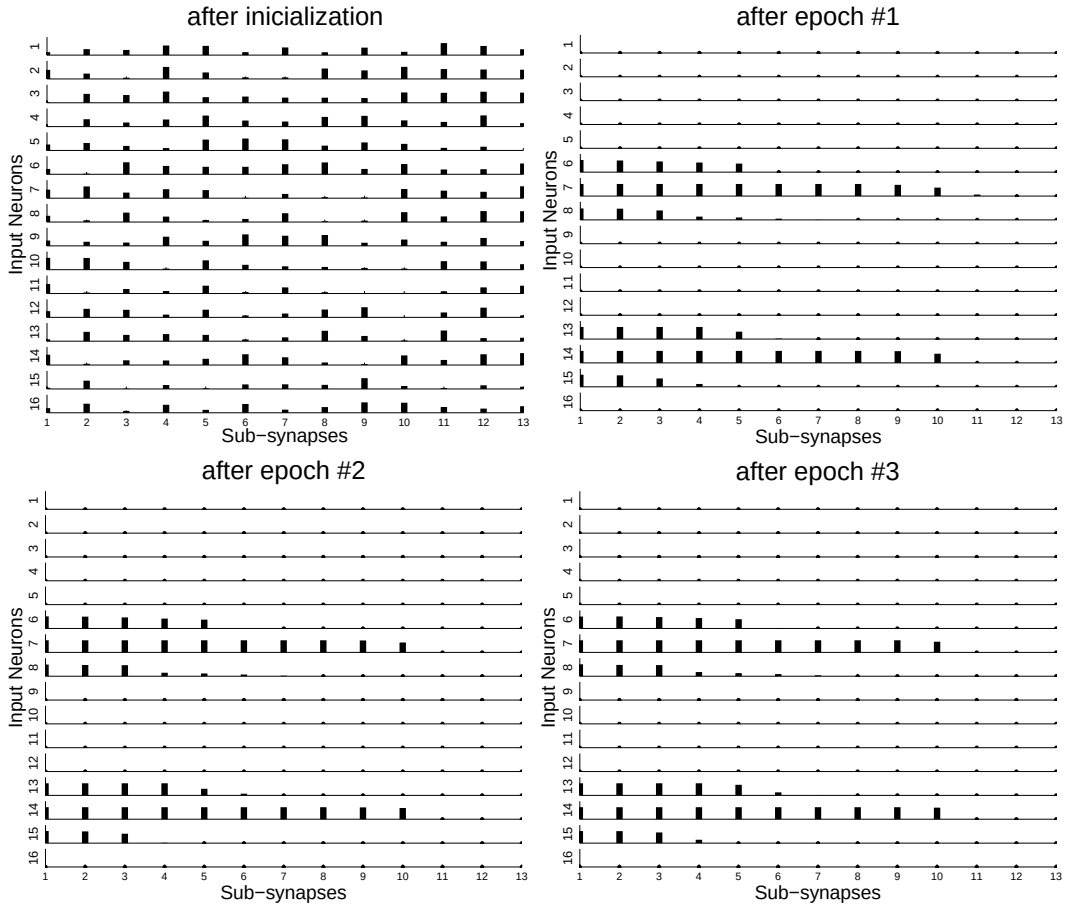

Figure 5.9: Weight evolution from the random initialization to the third and last training epoch.

The clusters were randomly generated with a total of 50 points each and, also randomly, 25 points were separated for training and the remaining points were used for validation. Each training epoch consisted of the presentation of the points and the weight updating according to equation (5.7). The experiments started with learning rates as low as $\eta = 0.0025$ and a number of epochs above 100, but the results demonstrated that higher learning rates and less epochs could be used without altering the correct separation of the clusters. In figure 5.9 it is possible to observe the weight evolution for the synapses connected to the output neuron 4 (see figure 5.10) using $\eta = 0.35$ and only 3 epochs.

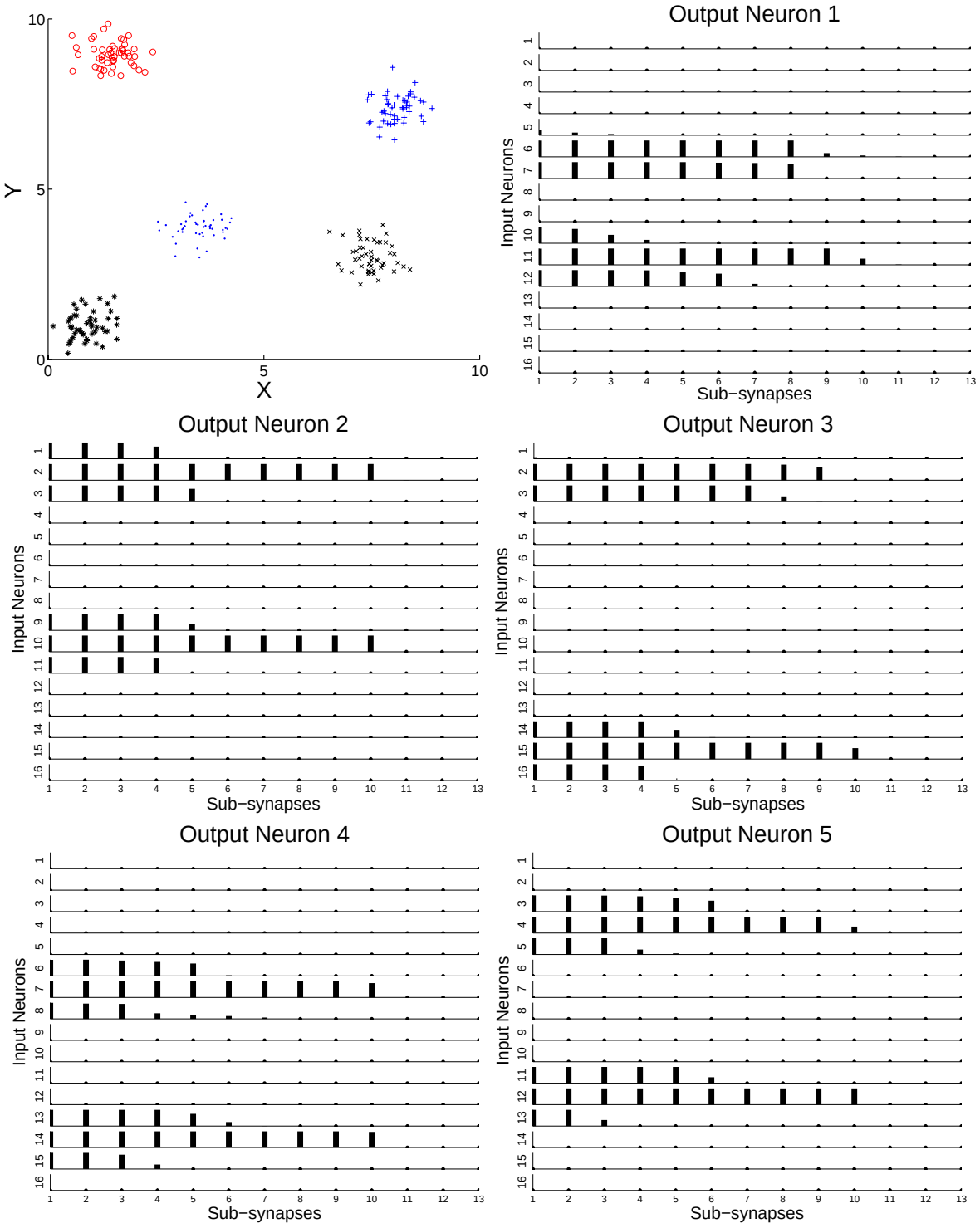

Figure 5.10: Five clusters correctly separated by a SNN with 8 neurons encoding each dimension. The resulting weights are shown for all five clusters. The SNN used 13 sub-synapses, with delays from 0 to 12ms and a learning rate $\eta = 0.35$.

Also important to the network performance are the limiar ϑ and the neighborhood ν, and if these parameters are not finely adjusted, the network will respond wrongly or will not respond at all to some of the inputs. The values used here were $\vartheta = 12$ and $\nu = 5$. The final results and corresponding weight configuration are shown in figure 5.10.

Other tests were also successfully made, with up to ten clusters, with minor necessary adjustments in the parameters. For the ten 50-points clusters 12 neurons were used to encode each dimension. The main parameters used are listed in table 5.1.

In order to demonstrate the capabilities of the present method with more complex clusters, we used a 256 x 256 pixels, 3-band RGB satellite image [2], where each color pixel is defined by a three dimensional vector with dimensions varying from 0 to 255. The network was trained using 32,769 randomly chosen points (of a total of the 65,536 points) to separate four clusters. The results of the clustering with the SNN are shown in figure 5.11 with the original image and the results of the same clustering using a self-organized map – SOM [BLdC00, Hay99] and the K-means algorithm. The SNN used only six neurons to encode each dimension. As expected, the clustering with the SNN showed the highest computational cost, followed by the SOM implementation, using a Matlab toolbox routine.

Table 5.1: Main parameters in the three examples of this chapter.

Example	Clusters	Dimensions	Points	ϑ	ν	Input	coding
1	5	2	250	12	5	16	10ms
2	10	2	500	12	8	24	10ms
3	4	3	65536	9	5	18	20ms

5.6 A Novel Approach

Although the existence of multiple synapses may be biologically plausible (e.g. [WZR98]), we can not help thinking of this case as being more a particularity than a rule. As a matter of fact, it is clear that the multiple synapse approach discussed here is far more a delay adaptation mechanism

[2]Extract of a IKONOS II scene, PSM mode, RGB bands 3-2-1 and Pan, natural colors, 1m resolution.

Figure 5.11: Results of an image clustering task. Clockwise from top: the original image, clustering by the k-means algorithm, clustering by the SOM (using Matlab) and clustering by the SNN.

than a solution to abide by biological plausibility. As a consequence, it is natural to suppose that there must be a way for single-synapse neurons to do the same sort of task of the multiple synapse neurons.

To test and demonstrate this novel approach, it is necessary to have a single EPSP equivalent to that produced by each multiple synapse assembly (figure 5.8). This equivalent EPSP was calculated using two methods, as shown in figure 5.12.

- In the first method, the equivalent EPSP was calculated using variable time constants (τ_{ij})

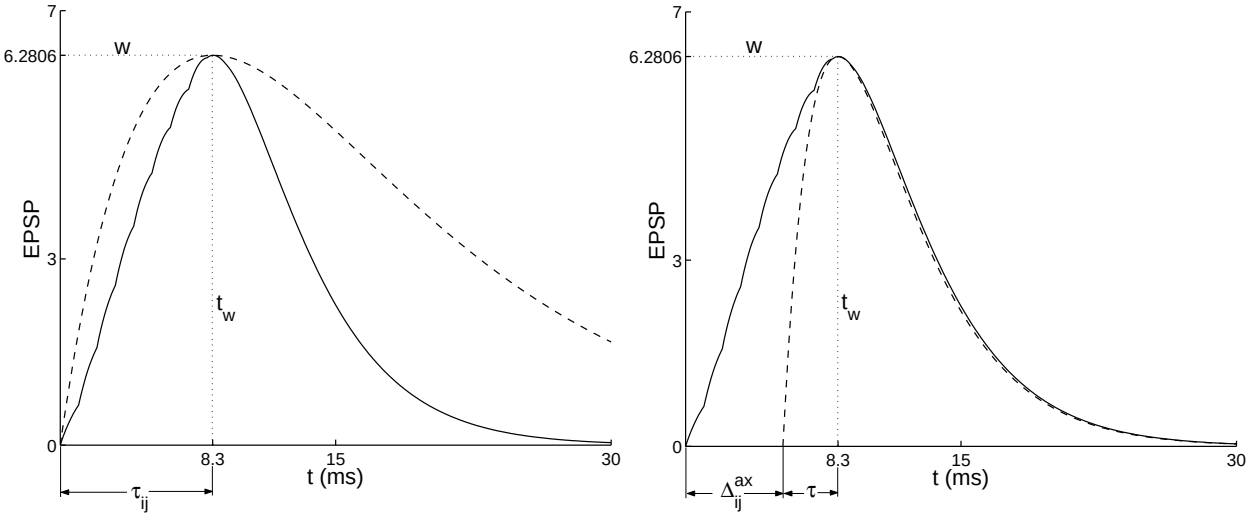

Figure 5.12: Two single-synapse EPSPs (dashed lines) equivalent to that produced by a multiple synapse assembly (solid lines). The figure shows the synapse between input neuron 1 and output neuron 2, in the five clusters experiment. At left, we have the equivalent EPSP using variable time constants and a null axonal delay (first method). At right we have the equivalent EPSP using a fixed time constant ($\tau = 3$) and a variable axonal delay (second method). Equation (2.3) was used in both cases.

equal to the instants of the maximum of the original EPSP, and a null axonal delay ($\Delta^{ax} = 0$).

- In the second method the equivalent EPSP was calculated using a fixed time constant ($\tau = 3$) and a variable axonal delay (Δ_{ij}^{ax}), so that the maximum of this EPSP coincides with the maximum of the original EPSP.

In both methods equation (2.3) was adopted for the equivalent EPSP, so that its maximum amplitude is the same as that of the original EPSP, occurring at the same time. The maximum value was used as the weight (figure 5.12).

Taking the five clusters example, for each of the 16 resultant EPSP we wrote down the maximum value and its time (w and t_w in figure 5.12), and calculated the single-synapse equivalent EPSP by both methods, and then repeated the clustering task with the same temporal encoded input. As expected, these new single-synapse SNNs presented the same results as the multiple synapse counterpart, except for the second method, where we observed five clustering errors (2%). The matrices for the weights, time constants and delays are shown below.

$w_{ij} =$

0	3.524	0.042	0	0
0	6.980	6.499	0	0
0	4.330	5.861	0	4.692
0	0	0	0	6.709
0.527	0	0	0	3.158
6.281	0	0	4.114	0
6.084	0	0	6.916	0
0	0	0	3.166	0
0	4.026	0	0	0
2.463	6.983	0	0	0
6.697	3.655	0	0	4.712
4.903	0	0	0	6.983
0	0	0	4.414	2.303
0	0	4.076	6.964	0
0	0	6.812	2.982	0
0	0	3.692	0	0

$\tau_{ij} =$

0	4.8	3	0	0
0	9.8	8.8	0	0
0	5.5	7.6	0	6.2
0	0	0	0	9.4
3.8	0	0	0	4.6
8.3	0	0	5.6	0
8	0	0	9.8	0
0	0	0	5.3	0
0	5.3	0	0	0
4.4	9.8	0	0	0
9.4	4.9	0	0	6
6.5	0	0	0	9.8
0	0	0	5.8	3.9
0	0	5.4	9.8	0
0	0	9.6	4.5	0
0	0	4.9	0	0

$\Delta_{ij}^{ax} =$

0	1.8	0	0	0
0	6.8	5.8	0	0
0	2.5	4.6	0	3.2
0	0	0	0	6.4
0.8	0	0	0	1.6
5.3	0	0	2.6	0
5	0	0	6.8	0
0	0	0	2.3	0
0	2.3	0	0	0
1.4	6.8	0	0	0
6.4	1.9	0	0	3
3.5	0	0	0	6.8
0	0	0	2.8	0.9
0	0	2.4	6.8	0
0	0	6.6	1.5	0
0	0	1.9	0	0

These results demonstrates the proposed approach but, albeit this successful outcome, a main question remains unanswered: how unsupervised reinforcement learning should be applied to such a SNN?

Despite some few references on delay adaptation may be found [EPE+99, EPE+00, LvH01, LKvH02, SZK99, YS98], most of these works only present a general approach, and none of them deal with the specific kind of problem we have here.

As a first trial to answer this remainig question, considering the results of the first method, we adopted the simultaneous and independent adaptation of the weights and time constants in a SNN of single synapses. For the weight adaptation we used the same learning function as in the previous examples, and for the adaptation of the time constants we used the learning function

mentioned in [IGE04, GK02b], defined by equation (5.8) and shown in figure 5.13.

$$L(\Delta t) = \begin{cases} e^{-\left(\frac{\Delta t + \alpha}{\tau}\right)} & \text{for } \Delta t \geq 0 \\ -e^{-\left(\frac{\Delta t + \alpha}{\tau}\right)} & \text{for } \Delta t < 0 \end{cases} \tag{5.8}$$

Although several variations and parameter set were used, this approach does not produced any acceptable results in almost all tentatives, suggesting that the independent adaptation of weights and time constants is not a good solution for the problem in perspective. As a matter of fact, the results of this approach showed more errors than those obtained just using a fixed value for the time constants and adapting only the weights. In this last case a few tentatives produced a clustering without errors.

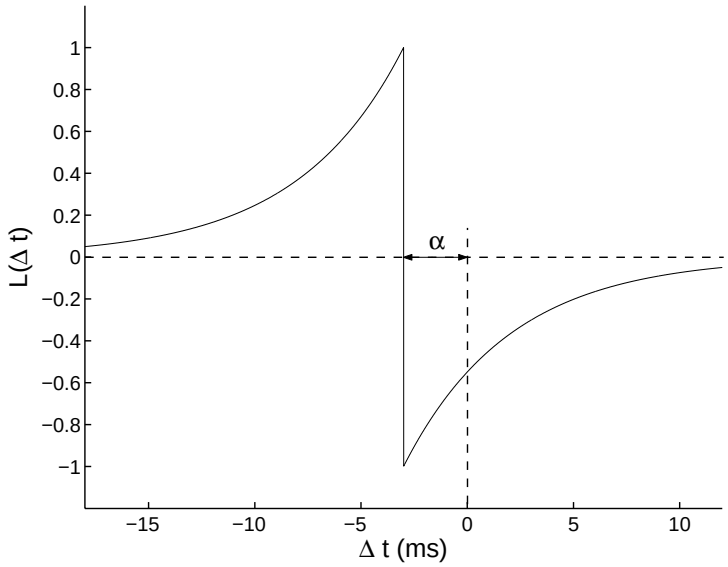

Figure 5.13: Alternative learning function applied to delay adaptation in a single synapse approach.

As a second trial, in order to find experimentally a relationship between weights and time constants, we plotted a map of the normalized weight and time constants values, for the five and ten clusters experiments. These maps, shown in figure 5.14, suggest that there may be indeed some relationship. A curve fitting procedure yielded equations (5.9) and (5.10), for the five and ten clusters cases, respectively.

$$\tau(w) = 0.2452e^{-0.07922w} + 0.09115e^{2.149w} \tag{5.9}$$

$$\tau(w) = 0.2716e^{-0.8521w} + 0.07901e^{2.413w} \tag{5.10}$$

Both experiments were then repeated 50 times each, using equations (5.9) and (5.10) to calculate the time constants as a function of the weights, which were adapted with the same procedures of the multiple synapse method, i.e., using equations (5.3) and (5.4). Around 80% of the tentatives showed no errors, indicating that there must be indeed a relationship connecting weights and time constants. The results also pointe out the need of a deeper theoretical and mathematical analsys to explore the complexity of the problem.

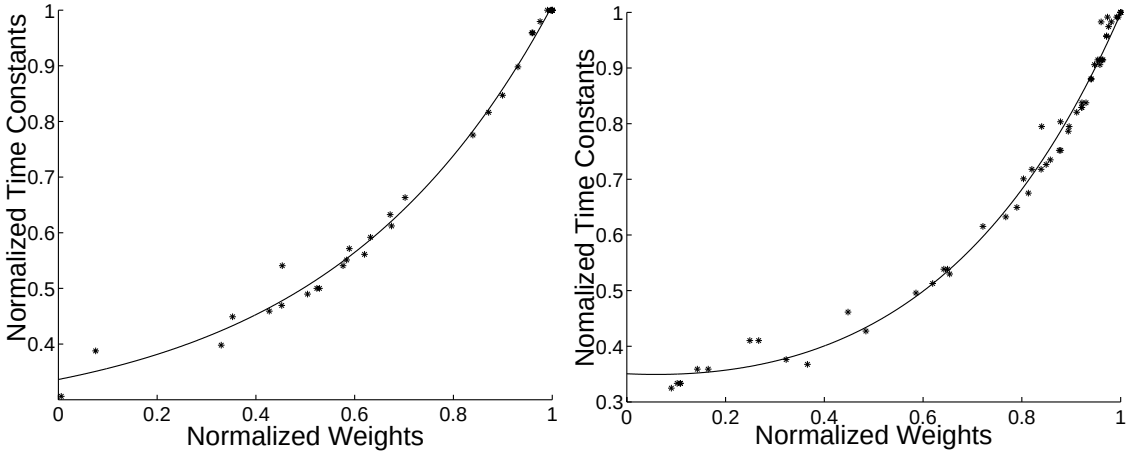

Figure 5.14: Maps for the five (left) and ten (right) clustering experiments to find experimentally a relationship between weights and time constants.

5.7 Conclusions

Together with some examples of the application of a multiple synapse SNN, we presented here a novel architecture with single synapses and applied it successfully to do clustering tasks.

Since the objective of the learning process is to approximate the firing times of all the neurons related to the same cluster, it is quite obvious that a neuron less stimulated (large Δt and thus, low weight) must have also a lower time constant, so it can fire faster and compensate for the large Δt. Similarly, a more stimulated neuron (small Δt and thus, high weight) must have also a higher time constant, so it can fire slower and compensate for the small Δt.

As already mentioned in the previous section, all the experimental results obtained in the development this chapter indicate that the simultaneous adaptation of weights and time constants (or axonal delays) must be submitted to a far more extensive theoretical analisys. Given the high complexity of the problem, it is not encompassed by the scope of the present work, and hence should be left to a further work.

Chapter 6

Conclusions

6.1 Outlook

In this work, as was our main target, we tried to summarize the foundations of spiking neurons and to emphasize the importance of considering this new paradigm when building neural networks. To accomplish this task we first presented the basic components of the nervous systems, giving a glimpse of all their underlaying complexity. Following, we then discussed some of the mathematical models that allow us to simulate these systems, from the simpler to the more complex ones. After the exposition of the theoretical basis, we briefly discussed the dynamics of real neural processes, with a few examples of application of the models previously described. It was also presented a practical applications of a neural network, built with more biologically inspired neurons, to perform what we could call *engineering* tasks. In this application we demonstrate how analog values can be temporally encoded and how a network can learn using this temporal code.

Even with these very short steps towards the realm of neuroscience, it is not difficult to realize how intricate things can get if we try to descend deeper into the details of neural simulation. However, this apparent difficulty should rather be regarded as an opportunity to use spike-timing as an additional variable in the information processing by neural networks [Boh04].

The construction presented here and also other types of SNN seem to be suitable for hardware implementation [MB98, SHY00, SSW+02, KA05]. There are already several implementations of Hebbian learning with spiking neurons [Häf00, LMB00]. Real spiking neural networks are

known to be very fast [Ger99, GK02b] and the single synapse approach discussed in Chapter 5 could represent a significant reduction in computation time if compared to the multiple synapse approach. It is worth noting that other learning methods have been used with SNN, like supervised learning [XE01, BKP02]. We can find also SNN aplications in other fields, e.g., [MB00, SC99].

In the field of real neural systems and cognitive sciences, the advance in computer power has ensued a trend to simulate large populations [IGE04, MG02, Vol04] to study their behavior. The elaboration and application of the more detailed neuron models to do engineering tasks is still in its beginning, and there is a lot of work to be done to make the present models more tractable.

Besides the extensive review of spiking neurons, although a definitive solution for the training problem is out of the scope of this work, we belive to be contibuting with a new direction of research by demonstrating that the single synapse approach can be adopted with the advantage of reducing the computational costs.

6.2 Future Works

Focusing specially in neural modeling and related applications, this work has only slightly touched the wide field of neuroscience, and so, it is only a first step to be followed by many others, toward different directions, representing the broad range of research possibilities for future development. For instance, one could go deeper into Neurophysiology to uncover the intricate structure of the nervous systems at the molecular level, enabling the use of proteins and other molecules for information-processing, computation in nonlinear media, computers based on physical reaction-diffusion systems found in chemical media, DNA computing, bioelectronics and protein-based optical computing, and biosensors.

Despite this broad range of possibilities, we will limit ourselves to only two research choices, directly related with the subjects discussed here: neural code and the single synapse approach suggested in Chapter 5. Although the code scheme used here to encode analog variables into precisely spike timing worked well in our implementation, this subject is worth of further investigation in order to determine how analog inputs are encoded in real nervous systems. It will not

be an easy task and this question still remains unsolved [GK02b].

For the single synapse approach proposed here, all the experimental results obtained indicate that the simultaneous adaptation of weights and time constants (or axonal delays, or both) should be submitted to a far more extensive theoretical analisys. We belive that axonal delays, time constants and weights are closely interrelated and, once a dependence relationship is determined, multiobjective optimization techniques could be employed to define a learning rule.

Appendix A

Matlab Scripts

We present here the final version of the Matlab 6 scripts [HL03] used in the sumulations of Chapter 5. The listed scripts were used to the five clusters experiment and can be applied to any number of cluster with little modifications.

A.1 Temporal encoding - Section 5.2

A.1.1 Linear encoding

```
function aus = kodieren_ln(ein, maxd, rho, plt)
%
% Funcao aus = kodieren_ln(ein, maxd, rho, plt)
%
% Descricao: Codifica valores analogicos transformando-os diretamente
%            em atrasos dentro do intervalo de codificacao. A quantidade
%            de neuronios e igual a quantidade de entradas (dimensoes).
%
%          ein = matriz i x j com os i exemplos para treinamento e
%                validacao e as j entradas analogicas (dimensoes).
%                Cada linha de ein e um conjunto de variaveis
%                analogicas de entrada.
%          aus = matriz m x n com os atrasos de todas as entradas
%                analogicas codificadas, onde m e a quantidade de
%                exemplos (m = i) e n e a quantidade total de
%                neuronios de entrada usados para codificar as
```

```
%                entradas analogicas. Cada linha de aus contem os
%                atrasos dos n neuronios de entrada (n = rf x j).
%         maxd = intervalo de codificacao.
%          rho = resolucao de tempo.
%          plt = 1 - plota a saida
%                0 - nao plota a saida
%
%-----------------------------------------
% Ultima alteracao em 09/04/2005 - RCBerredo
%-----------------------------------------
%
[t_cj,t_in]=size(ein);       % conjuntos (t_cj) e entradas analogicas (t_in)
%
% Normaliza entradas no intervalo (0,1)
for i=1:t_in
    range=max(ein(:,i))-min(ein(:,i));         % faixa de valores da variavel
    ein(:,i)=(ein(:,i)-min(ein(:,i)))./range;  % variaveis norm. de 0 a 1
end
%
% Calcula atrasos e monta saida
%aus=ein.*maxd;
aus=round((ein.*maxd)./rho).*rho;
%
% Plota os resultados
if plt==1
    figure;
    for i=1:t_cj
        for j=1:t_in
            subplot(t_in,1,j); hold on;
            H_line=plot([aus(i,j) aus(i,j)],[0 1],'b');
            set(H_line,'LineWidth',1);
            axis([0 maxd 0 2]);
            Ha_ax=gca;
            ylabel(sprintf('%1g',j),'FontSize',8);
            set(Ha_ax,'YTick',[]);
```

```
            set(Ha_ax,'XTick',[0 maxd]);
            %set(Ha_ax,'XTick',[0 maxd*0.2 maxd*0.4 maxd*0.6 maxd*0.8 maxd]);
            set(Ha_ax,'XGrid','on','XTickLabel',' ');
            if j==t_in
                set(Ha_ax,'XTickLabelMode','auto');
                xlabel('t (ms)','FontSize',12);
            end
            if j==1
                title(sprintf('Classes: %1g - Entradas: %1g - ...
                      Exemplos: %1g', t_in,t_in,t_cj),'FontSize',14);
            end
        end
    end
elseif plt~=0
    error('Parametro plt invalido: tem que ser 0 ou 1');
end
```

A.1.2 Receptive fields encoding

```
function aus = kodieren_rf(ein, rf, maxd, sigma, f_cut, rho, typ, plt)
%
% Funcao aus = kodieren_rf(ein, rf, maxd, sigma, f_cut, rho, typ, plt)
%
% Descricao: Codifica valores analogicos atraves de multiplos
%            campos receptivos fixos.
%
%          ein = matriz i x j com os i exemplos para treinamento e
%                validacao e as j entradas analogicas (dimensoes).
%                Cada linha de ein e um conjunto de variaveis
%                analogicas de entrada.
%          aus = matriz m x n com os atrasos de todas as entradas
%                analogicas codificadas, onde m e a quantidade de
%                exemplos (m = i) e n e a quantidade total de
%                neuronios de entrada usados para codificar as
%                entradas analogicas. Cada linha de aus contem os
%                atrasos dos n neuronios de entrada (n = rf x j).
```

```
%           rf = matriz linha com a quantidade de neuronios usados
%                na codificacao de cada entrada (dimensao).
%         maxd = intervalo de codificacao.
%        sigma = matriz linha com a largura de cada grupo de campos
%                receptivos. Recomenda-se o sigma = 1/(1,5(m-2)) para
%                o tipo 0 e sigma = 1/(1,5(m+1)) para o tipo 1.
%        f_cut = limite de disparo dos campos receptivos.
%          rho = resolucao de tempo.
%          typ = 1 - nehum neuronio esta localizado fora da faixa de
%                    valores a serem codificados.
%                0 - dois neuronios estao localizados fora da faixa de
%                    valores a serem codificados, sendo um de cada lado.
%          plt = 1 - plota a saida.
%                0 - nao plota a saida.
%
%-----------------------------------------
% Ultima alteracao em 17/05/2005 - RCBerredo
%-----------------------------------------
%
t_neur=sum(rf);              % total de neuronios
[t_cj,t_in]=size(ein);       % conjuntos (t_cj) e entradas analog. (t_in)
aus=zeros(t_cj,t_neur);      % cria matriz de saida
%sigma=0.233./rf;             % valor para que o atraso max < 0.9*dmax
%
% Processa as entradas
n=0;   % controle de colunas da matriz de saida (aus)
for i=1:t_in
    ct=zeros(1,rf(i));      % centros das curvas
%
% Verifica quantidade de neuronios
    if rf(i)<3
        error('Quantidade de campos receptivos tem que ser maior que 3');
    end
% Normaliza entradas no intervalo (0,1)
    range=max(ein(:,i))-min(ein(:,i));        % faixa de valores da variavel
```

```
    ein(:,i)=(ein(:,i)-min(ein(:,i)))./range; % variaveis norm. de 0 a 1
% Define os centros das curvas
    if typ==1
        cdis=1/rf(i);          % distancia entre os centros
        for j=1:rf(i)
            ct(j)=(j-0.5)*cdis;
        end
    elseif typ==0
        cdis=1/(rf(i)-2);      % distancia entre os centros
        for j=1:rf(i)
            ct(j)=(j-1.5)*cdis;
        end
    else
        error('Parametro typ invalido: tem que ser 0 ou 1');
    end
% Calcula atrasos e monta saida
    for j=1:t_cj
        for k=1:rf(i)
            aux=maxd-maxd.*exp(-((ein(j,i)-ct(k))^2)./(2.*sigma(i)^2));
            aus(j,k+n)=round(aux*(1/rho))*rho;  % arredonda
            if aus(j,k+n)>maxd*f_cut   % se o neuronio nao dispara
                aus(j,k+n)=-1;         % o atraso e -1
            end
        end
    end
    n=n+rf(i);   % controle de colunas da matriz de saida (aus)
end
%
% Plota os resultados
if plt==1
    figure;
    for i=1:t_cj
        for j=1:t_neur
            subplot(t_neur,1,j); hold on;
            H_line=plot([aus(i,j) aus(i,j)],[0 1],'b');
```

```
            set(H_line,'LineWidth',1);
            axis([0 maxd 0 2]);
            Ha_ax=gca;
            ylabel(sprintf('%1g',j),'FontSize',8);
            set(Ha_ax,'YTick',[]);
           set(Ha_ax,'XTick',[0 maxd]);
            %set(Ha_ax,'XTick',[0 maxd*0.2 maxd*0.4 maxd*0.6 maxd*0.8 maxd]);
           set(Ha_ax,'XGrid','on','XTickLabel',' ');
           if j==t_neur
               set(Ha_ax,'XTickLabelMode','auto');
               xlabel('t (ms)','FontSize',12);
           end
           if j==1
               title(sprintf('Classes: %1g - Entradas: %1g - ...
                     Exemplos: %1g', t_in,t_neur,t_cj),'FontSize',14);
              end
         end
      end
    figure; hold on;
    x=0:0.01:1;
    for i=1:rf(end)
        y=maxd.*exp(-(x-ct(i)).^2./(2*sigma(end)^2));
        plot(x,y);
    end
    plot([0 1],[maxd*(1-f_cut)  maxd*(1-f_cut)],'r');
elseif plt~=0
    error('Parametro plt invalido: tem que ser 0 ou 1');
end
```

A.1.3 Simple clustering example

```
% cl_spk_2c - teste de classificacao com SNN
%
% Ultima alteracao 05/06/2005 - RCBerredo
%
% Usa 2 conjuntos com distribuicao gaussiana com media zero.
```

```
% Cada conjunto tem 50 pontos.
% O metodo usado foi o de adaptar, de forma supervisionada,
% os atrasos axonais. Os pesos foram mantidos iguais a 1.
%
% Classes (saidas): 2
% Codificacao: Linear direta com um neuronio de referencia
% Conj. de Campos Receptivos: 0
% Tipo de Sinapses: simples
%
%-------------------------------------------------------------------------
% Prepara Entrada
aux1=dlmread('cj_spk2a_orig.txt');   % dados de entrada
%-------------------------------------------------------------------------
% Define Parametros Iniciais
tam=size(aux1,1);               % quantidade de conjuntos
in_neu=size(aux1,2)+1;          % qtde de neuronios de entrada
out_neu=2;                      % qtde de neuronios na saida (classes)
rho=0.1;                        % resolucao de tempo
tau=1.84;   %1.93;                          % cte de tempo do EPSP
maxd=10;                        % intervalo de codificacao
tmax=30;                        % maximo intervalo de calculo
teta=2.04;  %2.35;
%-------------------------------------------------------------------------
% Treina a rede
tra=50;   trb=(tam/2)+tra;   % tamanho do conjunto de treinamento
aus=kodieren_ln(aux1,maxd,rho,0);            % codifica entradas
%ca=[2.0 2.5];%ca=[7.8 6.5];
ca=round((sum(aus(1:tra,:))./tra)./rho).*rho;  % calcula media da classe 1
%cb=[8.2 6.4];%cb=[3 3.2];
cb=round((sum(aus(51:trb,:))./tra)./rho).*rho; % calcula media da classe 2
d=maxd-[ca(1) cb(1); ca(2) cb(2); maxd maxd];  % atrasos dos centros
w=[1 1;1 1;1 1];%[1 1.42;1 1.38;1 1];
%-------------------------------------------------------------------------
% Testa a rede
class=zeros(1,tam);                          % cria tabela de classes
```

```
aux1=dlmread('cj_spk2a.txt');                % dados de entrada
aus=kodieren_ln(aux1,maxd,rho,0);           % codifica entradas
aus=[aus ones(tam,1).*maxd];             % acrescenta referencia
for k=1:tam
    t=0;%min(aus(k,:));
    neu=0;
    while neu==0 & t<=tmax
        out=zeros(1,out_neu);
        for j=1:out_neu
            for i=1:in_neu
                dt=t-aus(k,i);
                sai=w(i,j)*((dt-d(i,j))/tau)*exp(1-((dt-d(i,j))/tau));
                if sai<0
                    sai=0;
                end
                out(j)=out(j)+sai;
            end
            %subplot(out_neu,1,j); hold on;
            %plot(t,out(j),'.',t,teta,'r.'); drawnow;
        end
        if max(out)>=teta
            neu=find(out==max(out));
            inst=t;
        end
        t=t+rho;
    end
    if size(neu,2)>1    % se deu empate...
        neu=0;          % classe = 0
    end
    if neu==0           % se acabou limite de tempo
       class(k)=0;      % classe = 0
    end
    class(k)=neu;       % guarda a classe
    %pause
    %h=gcf;              % pega hadle da figura
```

```
    %close(h);           % fecha figura
end
%-------------------------------------------------------------------------
% Plota resultado do teste
ptos={'+k' 'ok' 'or' '+b' 'xk' 'sr' 'db'...
      '.k' '*r' 'ob' '+k' 'xr' 'sb' 'dk'...
      '.r' '*b' 'ok' '+r' 'xb' 'sk' 'dr'};
figure; hold on;
for i=1:out_neu
    a=find(class==i);
    for n=1:size(a,2)
        plot(aux1(a(n),1),aux1(a(n),2),char(ptos(i)));
    end
end
a=find(class==0);
for n=1:size(a,2)
    plot(aux1(a(n),1),aux1(a(n),2),char(ptos(out_neu+1)));
end
%-------------------------------------------------------------------------
%function plota(p1, p2, px, py, teta)
%    subplot(p1,1,p2); hold on;
%    plot(px,py,'.',px,teta,'r.'); drawnow;
%-------------------------------------------------------------------------
```

A.2 Multiple synapse architecture - Section 5.5

A.2.1 Clustering

```
% cl_spk_5a - teste de classificacao com SNN
%
% Ultima alteracao 19/07/2005 - RCBerredo
%
% Usa 5 conjuntos com distribuicao gaussiana com media zero.
% Cada conjunto tem 50 pontos.
%
% Classes (saidas): 5
```

```
% Codificacao: Campos receptivos fixos
% Conj. de Campos Receptivos: 1
% Tipo de Sinapses: multiplas
%
%-------------------------------------------------------------------------
% Prepara Entrada
clear all;
aux1=dlmread('cj_spk5.txt');   % dados de entrada
%-------------------------------------------------------------------------
% Define Parametros Iniciais
out_neu=5;                      % qtde de neuronios na saida (classes)
rho=0.1;                        % resolucao de tempo
conj=125;                        % qtde de conjuntos de treinamento
tau=3;                          % cte de tempo do EPSP
rf=[8 8];                       % qtde de neuronios por entrada
sigma=[1/(1.5*(rf(1)-2)) 1/(1.5*(rf(2)-2))];   % largura dos campos recep.
f_cut=0.9;
maxd=10;                        % intervalo de codificacao
%-------------------------------------------------------------------------
% Define Parametros de Aprendizado
d=[0 1 2 3 4 5 6 7 8 9 10 11 12]; % atrasos das sub-sinapses
w_max=1; w_min=0;               % pesos maximo e minimo
max_epoch=3;                   % no. maximo de epocas
t_max=30;                       % tempo maximo de treinamento
eta=0.35;                      % taxa de aprendizado
beta=0.2;                       % cte para janela de aprendizado
nu=5.0;                         % vizinhanca
dx=2.3;                         % deslocamento da janela de aprendizado
                                % p/ esquerda (+) ou direita (-)
%-------------------------------------------------------------------------
% Calcula Parametros
kapa=1-(nu^2)/(2*log(beta/(beta+1))); % cte para janela de aprendizado
in_neu=sum(rf);                       % qtde de neuronios na entrada
ssin=size(d,2);                       % qtde de sub-sinapses
%teta=1.5*(in_neu/out_neu)*(w_max-w_min)*ssin/2;
```

```
teta=12;                          % valor do limiar
%-------------------------------------------------------------------------
% Inicializa Pesos
w=zeros(in_neu,out_neu,ssin);
for i=1:in_neu
    for j=1:out_neu
        w(i,j,:)=w_min+rand(1,ssin).*(w_max-w_min);
    end
end
%-------------------------------------------------------------------------
% Inicia Treinamento
tic;
aus=kodieren_rf(aux1,rf,maxd,sigma,f_cut,rho,0,0);  % codifica entrada
ctrl_1=1;                                          % controle do loop
delta_t=zeros(1,1,ssin);
h=waitbar(ctrl_1/max_epoch,sprintf('Executando %i de %i iteracoes',...
  ctrl_1, max_epoch));
%show_w(w);
f_times=zeros(2,1);
while ctrl_1<=max_epoch
    for z=1:conj
        tr=aus(z,:);
        [neu, inst]=wer_schiesst(tr,t_max,w,d,tau,rho,teta,0);
        f_times=[f_times [neu;inst]];
        if neu~=0 & size(neu,2)==1
            for i=1:in_neu
                if tr(i)==-1
                    w(i,neu,:)=w(i,neu,:)-eta;
                else
                    delta_t(1,1,:)=tr(i)+d-inst;  % calcula delta_t
                    w(i,neu,:)=w(i,neu,:)+eta.*((1+beta).*...
                             exp(-(((delta_t+2.3).^2)./(2*kapa-2)))-beta);
                end
                ndx=find(w(i,neu,:)<w_min);   % garante limites
                for u=1:size(ndx,1)
```

```
                    w(i,neu,ndx(u))=w_min;
                end
                ndx=find(w(i,neu,:)>w_max);   % garante limites
                for u=1:size(ndx,1)
                    w(i,neu,ndx(u))=w_max;
                end
            end
        end
    end
    ctrl_1=ctrl_1+1;                    % incrementa contador de epocas
    waitbar(ctrl_1/max_epoch,h,sprintf('Executando %i de %i iteracoes',...
    ctrl_1, max_epoch));
    teta=teta+(0.3*teta)/max_epoch;
    %show_w(w);
end
close(h);
toc;
%-------------------------------------------------------------------------
% Testa a rede
[conj,in_neu]=size(aus);
class=zeros(1,conj);            % cria tabela de classes
for i=1:conj
    tr=aus(i,:);                % conjunto de teste
    [neu,inst]=wer_schiesst(tr,t_max,w,d,tau,rho,teta,0); % quem disparou?
    if size(neu,2)>1    % se deu empate...
        neu=0;          % classe = 0
    end
    class(i)=neu;                           % guarda a classe
end
%-------------------------------------------------------------------------
% Plota resultado do teste
ptos={'.b' '*k' 'or' '+b' 'xk' 'sr' 'db'...
      '.k' '*r' 'ob' '+k' 'xr' 'sb' 'dk'...
      '.r' '*b' 'ok' '+r' 'xb' 'sk' 'dr'};
figure; hold on;
```

```
for i=1:out_neu
    a=find(class==i);
    for n=1:size(a,2)
        plot(aux1(a(n),1),aux1(a(n),2),char(ptos(i)));
    end
end
a=find(class==0);
for n=1:size(a,2)
    plot(aux1(a(n),1),aux1(a(n),2),char(ptos(out_neu+1)));
end
```

A.2.2 Image clustering - K-means

```
% cl_img_kmeans - Classifica imagem com k-means
%
% o nome do arquivo com a imagem e 'original_corte.jpg'
% este script depende da funcao Kmeans_var()
%
% Ultima alteracao 24/07/2005 - RCBerredo
%
clear all
% le o arquivo e transforma de 256 x 256 x 3 para 65536 x 3
imag=double(imread('original_corte.jpg'));  % transforma em double para
                                            % poder processar
[x1 y1 z1]=size(imag);             % pega dimensoes
% transforma array tri para bi-dimensional x1 x y1 x z1 --> (x1 * y1) x z1
imagx=zeros((x1*y1),z1); conta=1; % cria area de saida
for i=1:x1
    for j=1:y1
        for k=1:z1
            imagx(conta,k)=imag(i,j,k);
        end
    conta=conta+1;
    end
end
%classifica
```

```
[c,u,saida,class] = Kmeans_var(imagx, 4, 500, 0.01);
c_int=round(c);    % os centros devem ser numeros inteiros
imagy=uint8(zeros((x1*y1),z1));  % cria area de saida unsigned int
% substitui os vetores pelos centros das classes
for i=1:(x1*y1)
    imagy(i,:)=c_int(class(i),:);
end
% Volta ao array tri-dimensional
imagz=uint8(zeros(x1,y1,z1)); conta=1;
for i=1:x1
    for j=1:y1
        for k=1:z1
            imagz(i,j,k)=imagy(conta,k);
        end
    conta=conta+1;
    end
end
% salva a imagem em .bmp
imwrite(imagz,'saida_kmeans.jpg','jpg','Quality',100);
% plota os pontos
r=imagx(:,1);
g=imagx(:,2);
b=imagx(:,3);
figure(1); plot3(r,g,b,'.'); grid on;
box on; axis tight;
title('Points Distribution','FontSize',14);
xlabel('Red','FontSize',14);
ylabel('Green','FontSize',14);
zlabel('Blue','FontSize',14);
% plota os centros
cr=c(:,1);
cg=c(:,2);
cb=c(:,3);
figure(2); hold on; grid on;
box on; axis([0 256 0 256 0 256]);
```

```
plot3(cr,cg,cb,'r*');
plot3([0; 255],[0; 255],[0; 255],'r');
title('Center Location','FontSize',14);
xlabel('Red','FontSize',14);
ylabel('Green','FontSize',14);
zlabel('Blue','FontSize',14);
```

A.2.3 Image clustering - SOM

```
% cl_img_som - Classifica imagem com SOM
%
% o nome do arquivo com a imagem e 'original_corte.jpg'
%
% Ultima alteracao 24/07/2005 - RCBerredo
%
% le o arquivo e transforma de 256 x 256 x 3 para 65536 x 3
imag=double(imread('original_corte.jpg'));  % transforma em double
                                            % para poder processar
[x1 y1 z1]=size(imag);             % pega dimensoes
% transforma array tri para bi-dimens. x1 x y1 x z1 -> (x1 * y1) x z1
imagx=zeros((x1.*y1),z1); conta=1; % cria area de saida
for i=1:x1
    for j=1:y1
        for k=1:z1
            imagx(conta,k)=imag(i,j,k);
        end
    conta=conta+1;
    end
end
% monta conjunto de treinamento
treina=[];
for i=1:8:(x1*y1)
    treina=[treina; imagx(i,:)];
end
% monta rede SOM e treina
net=newsom([0 255; 0 255; 0 255], [1 4], 'gridtop');
```

```
net.trainParam.epochs = 50;
net = train(net,treina');
% plotsom(net.iw{1,1},net.layers{1}.distances);
c_int=uint8(round(net.iw{:,:}));  % pega os centros e arredonda p/ int
c=double(c_int);
% classifica
class=vec2ind(sim(net,imagx'));     % monta matriz com os indices
imagy=uint8(zeros((x1.*y1),z1));    % cria area de saida unsigned int
% substitui os vetores pelos centros das classes
for i=1:(x1*y1)
    imagy(i,:)=c_int(class(i),:);
end
% Volta ao array tri-dimensional
imagz=uint8(zeros(x1,y1,z1)); conta=1;
for i=1:x1
    for j=1:y1
        for k=1:z1
            imagz(i,j,k)=imagy(conta,k);
        end
    conta=conta+1;
    end
end
% salva a imagem em .bmp
imwrite(imagz,'saida_som.jpg','jpg','Quality',100);
% plota os pontos
r=imagx(:,1);
g=imagx(:,2);
b=imagx(:,3);
figure(1); plot3(r,g,b,'.'); grid on;
box on; axis tight;
title('Points Distribution','FontSize',14);
xlabel('Red','FontSize',14);
ylabel('Green','FontSize',14);
zlabel('Blue','FontSize',14);
% plota os centros
```

```
cr=c(:,1);
cg=c(:,2);
cb=c(:,3);
figure(2); hold on; grid on;
box on; axis([0 256 0 256 0 256]);
plot3(cr,cg,cb,'r*');
plot3([0; 255],[0; 255],[0; 255],'r');
title('SOM Center Location','FontSize',14);
xlabel('Red','FontSize',14);
ylabel('Green','FontSize',14);
zlabel('Blue','FontSize',14);
```

A.2.4 Image clustering - SNN

```
% cl_img_snn - Classifica imagem com SNN
%
% Ultima alteracao 24/07/2005 - RCBerredo
%
% O nome do arquivo com a imagem e 'original_corte.jpg'
% Classes (saidas): 3
% Codificacao: Campos receptivos fixos
% Conj. de Campos Receptivos: 1
% Tipo de Sinapses: multiplas
%--------------------------------------------------------------------------
clear all
%
% le o arquivo e transforma em double para poder processar
aux=double(imread('original_corte.jpg')); % arquivo de imagem
[x1 y1 z1]=size(aux);                     % pega dimensoes
%--------------------------------------------------------------------------
% transforma array tri para bi-dimensional x1 x y1 x z1 --> (x1 * y1) x z1
aux1=zeros((x1*y1),z1); conta=1;          % aux1 = entradas analogicas
for i=1:x1
    for j=1:y1
        for k=1:z1
            aux1(conta,k)=aux(i,j,k);
```

```
        end
    conta=conta+1;
    end
end
%-------------------------------------------------------------------------
% Define Parametros Iniciais
out_neu=4;                      % qtde de neuronios na saida (classes)
rho=0.1;                        % resolucao de tempo
conj=32768; % !!!!              % qtde de conjuntos de treinamento
tau=3;                          % cte de tempo do EPSP
rf=[6 6 6];                     % qtde de neuronios por entrada
% largura dos campos receptivos
sigma=[1/(1.5*(rf(1)-2)) 1/(1.5*(rf(2)-2)) 1/(1.5*(rf(3)-2))];
f_cut=0.9;
maxd=20;                        % intervalo de codificacao
%-------------------------------------------------------------------------
% Define Parametros de Aprendizado
d=[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]; % atrasos das sub-sinapses
w_max=1; w_min=0;               % pesos maximo e minimo
max_epoch=3;                    % no. maximo de epocas
t_max=30;                       % tempo maximo de treinamento
eta=0.35;                       % taxa de aprendizado
beta=0.2;                       % cte para janela de aprendizado
nu=5.0;                         % vizinhanca
%-------------------------------------------------------------------------
% Calcula Parametros
kapa=1-(nu^2)/(2*log(beta/(beta+1))); % cte para janela de aprendizado
in_neu=sum(rf);                       % qtde de neuronios na entrada
ssin=size(d,2);                       % qtde de sub-sinapses
teta=9.0;                             % valor do limiar
%-------------------------------------------------------------------------
% Inicializa Pesos
w=zeros(in_neu,out_neu,ssin);
for i=1:in_neu
    for j=1:out_neu
```

```
        w(i,j,:)=w_min+rand(1,ssin).*(w_max-w_min);
    end
end
%--------------------------------------------------------------------------
% Inicia Treinamento
aus=kodieren_rf(aux1,rf,maxd,sigma,f_cut,rho,0,0);  % codifica entrada
ctrl_1=1;                                           % controle do loop
delta_t=zeros(1,1,ssin);
h=waitbar(ctrl_1/max_epoch,sprintf('Executando %i de %i iteracoes',...
  ctrl_1, max_epoch));
while ctrl_1<=max_epoch
    for z=1:conj
        tr=aus(z,:);
        [neu, inst]=wer_schiesst(tr,t_max,w,d,tau,rho,teta,0);
        if neu~=0 & size(neu,2)==1
            for i=1:in_neu
                if tr(i)==-1
                    w(i,neu,:)=w(i,neu,:)-eta;
                else
                    delta_t(1,1,:)=tr(i)+d-inst;  % calcula delta_t
                    w(i,neu,:)=w(i,neu,:)+eta.*((1+beta).*...
                             exp(-(((delta_t+2.3).^2)./(2*kapa-2)))-beta);
                end
                ndx=find(w(i,neu,:)<w_min);   % garante limites
                for u=1:size(ndx,1)
                    w(i,neu,ndx(u))=w_min;
                end
                ndx=find(w(i,neu,:)>w_max);   % garante limites
                for u=1:size(ndx,1)
                    w(i,neu,ndx(u))=w_max;
                end
            end
        end
    end
    ctrl_1=ctrl_1+1;                    % incrementa contador de epocas
```

```
    waitbar(ctrl_1/max_epoch,h,sprintf('Executando %i de %i iteracoes',...
    ctrl_1, max_epoch));
    teta=teta+(0.3*teta)/max_epoch;
end
close(h);
%-------------------------------------------------------------------------
% Define as classes com a SNN treinada
[conj,in_neu]=size(aus);
class=zeros(1,conj);            % cria tabela de classes
for i=1:conj
    tr=aus(i,:);                % conjunto de teste
    [neu,inst]=wer_schiesst(tr,t_max,w,d,tau,rho,teta,0); % quem disparou?
    if size(neu,2)>1    % se deu empate...
        neu=out_neu+1;  % classe = n de classes+1
    end
    class(i)=neu;                           % guarda a classe
end
% calcula os centros das classes
c=zeros(out_neu+1,size(rf,2));      % cada linha e um centro
for i=1:out_neu
    a=find(class==i);
    for n=1:size(a,2)
        c(i,1)=c(i,1)+aux1(a(n),1);
        c(i,2)=c(i,2)+aux1(a(n),2);
        c(i,3)=c(i,3)+aux1(a(n),3);
    end
    c(i,:)=c(i,:)./size(a,2);
end
%a=find(class==out_neu+1);
%for n=1:size(a,2)
%    c(out_neu+1,:)=[256 0 0];  % pinta de vermelho as classificacoe erradas
%end
c_int=round(c);           % os centros devem ser numeros inteiros
imagy=uint8(zeros((x1*y1),z1));  % cria area de saida unsigned int
for i=1:(x1*y1)           % substitui os vetores pelos centros das classes
```

```
    imagy(i,:)=c_int(class(i),:);
end
imagz=uint8(zeros(x1,y1,z1)); conta=1; % volta ao array tri-dimensional
for i=1:x1
    for j=1:y1
        for k=1:z1
            imagz(i,j,k)=imagy(conta,k);
        end
    conta=conta+1;
    end
end
imwrite(imagz,'saida_snn.jpg','jpg','Quality',100); % salva a imagem em .bmp
% plota os pontos
r=aux1(:,1);
g=aux1(:,2);
b=aux1(:,3);
figure(1); plot3(r,g,b,'.'); grid on;
box on; axis tight;
title('Points Distribution','FontSize',14);
xlabel('Red','FontSize',14);
ylabel('Green','FontSize',14);
zlabel('Blue','FontSize',14);
% plota os centros
cr=c(:,1);
cg=c(:,2);
cb=c(:,3);
figure(2); hold on; grid on;
box on; axis([0 256 0 256 0 256]);
plot3(cr,cg,cb,'r*');
plot3([0; 255],[0; 255],[0; 255],'r');
title('SNN Center Location','FontSize',14);
xlabel('Red','FontSize',14);
ylabel('Green','FontSize',14);
zlabel('Blue','FontSize',14);
```

A.3 Single synapse architecture - Section 5.6

A.3.1 First method - Time constant adaptation

```
% single_syn_5a - teste de classificacao com SNN
%
% Ultima alteracao 07/08/2005 - RCBerredo
%
% Testa equivalente de sinapse simples. Transforma as sinapses
% multiplas em simples. Usa o resultado de cl_spk_5a.m com 16
% neuronios de entrada e cinco de saida.
%
% Cada conjunto tem 50 pontos.
%
% Classes (saidas): 5
% Codificacao: Campos receptivos fixos
% Conj. de Campos Receptivos: 1
% Tipo de Sinapses: simples
% Atrasos: constante de tempo do EPSP
%
%--------------------------------------------------------------------------
% Prepara Entrada
clear all;
load('single_syn.mat');        % carrega vetores de pesos w
% Em single_syn.mat estao:
%   aux1 = matriz 250x2 com as entradas analogicas
%    aus = matriz 250x16 com as entradas anaogicas codificadas
%     dx = valor de tau (instantes em que o EPSP atinge o maximo)
%     wx = pesos (valores maximos do EPSP)
%   epsp = valores dos epsp resultantes para cada uma das 80 (16x5)
%          sinapses multiplas durante 30ms em intervalos de 0.1ms.
%          As primeiras 16 linhas referem-se ao neuronio de saida 1,
%          depois as proximas 16 ao neuronio 2, etc. Desta matriz
%          foram extraidos dx e wx.
%--------------------------------------------------------------------------
% Define Parametros Iniciais
```

```
[in_neu out_neu]=size(dx);      % qtde de entradas e saidas
conj=size(aus,1);               % qtde de conjuntos a classificar
rho=0.1;                        % resolucao de tempo
teta=12;
tmax=30;
%-------------------------------------------------------------------------
% Inicia Classificacao
class=zeros(1,conj);            % cria tabela de classes
for n=1:conj
    in=aus(n,:);
    ndx=find(in~=-1);           % acha as entradas que disparam
    tam=size(ndx,2);
    t=0;                        % comeca em t=0
    neu=0;
%    h=figure; hold on;
    while neu==0 & t<=tmax
        out=zeros(1,out_neu);
        for j=1:out_neu         % para cada saida
            for i=1:tam
                if wx(ndx(i),j)==0 | dx(ndx(i),j)==0
                    sai=0;
                else
                    dt=(t-in(ndx(i)))/dx(ndx(i),j);
                    sai=wx(ndx(i),j)*dt*exp(1-dt);
                    if sai<0
                       sai=0;
                    end
                end
                out(j)=out(j)+sai;
            end
%            plot(t,out(j),t,teta);
        end
        if max(out)>=teta
            neu=find(out==max(out));
            inst=t;
```

```
        end
        t=t+rho;
    end
%    pause; close(h);
    if size(neu,2)>1    % se deu empate...
        neu=0;          % classe = 0
    end
    class(n)=neu;                          % guarda a classe
end
%-------------------------------------------------------------------------
% Plota resultado do teste
ptos={'.b' '*k' 'or' '+b' 'xk' 'sr' 'db'...
      '.k' '*r' 'ob' '+k' 'xr' 'sb' 'dk'...
      '.r' '*b' 'ok' '+r' 'xb' 'sk' 'dr'};
figure; hold on;
for i=1:out_neu
    a=find(class==i);
    for n=1:size(a,2)
        plot(aux1(a(n),1),aux1(a(n),2),char(ptos(i)));
    end
end
a=find(class==0);
for n=1:size(a,2)
    plot(aux1(a(n),1),aux1(a(n),2),char(ptos(out_neu+1)));
end
```

A.3.2 Second method - Axonal delay adaptation

```
% single_syn_5b - teste de classificacao com SNN
%
% Ultima alteracao 15/08/2005 - RCBerredo
%
% Testa equivalente de sinapse simples. Transforma as sinapses
% multiplas em simples. Usa o resultado de cl_spk_5a.m com 16
% neuronios de entrada e cinco de saida.
%
```

```
% Cada conjunto tem 50 pontos.
%
% Classes (saidas): 5
% Codificacao: Campos receptivos fixos
% Conj. de Campos Receptivos: 1
% Tipo de Sinapses: simples
% Atrasos: atraso axonal
%
%-------------------------------------------------------------------------
% Prepara Entrada
clear all;
load('single_syn.mat');        % carrega vetores de pesos w
% Em single_syn.mat estao:
%   aux1 = matriz 250x2 com as entradas analogicas
%    aus = matriz 250x16 com as entradas anaogicas codificadas
%     dx = valor de tau (instantes em que o EPSP atinge o maximo)
%     cx = dx-3
%     wx = pesos (valores maximos do EPSP)
%   epsp = valores dos epsp resultantes para cada uma das 80 (16x5)
%          sinapses multiplas durante 30ms em intervalos de 0.1ms.
%          As primeiras 16 linhas referem-se ao neuronio de saida 1,
%          depois as proximas 16 ao neuronio 2, etc. Desta matriz
%          foram extraidos dx e wx.
%-------------------------------------------------------------------------
% Define Parametros Iniciais
[in_neu out_neu]=size(dx);      % qtde de entradas e saidas
conj=size(aus,1);               % qtde de conjuntos a classificar
rho=0.1;                        % resolucao de tempo
teta=12;
tmax=30;
%-------------------------------------------------------------------------
% Inicia Classificacao
class=zeros(1,conj);            % cria tabela de classes
for n=1:conj
    in=aus(n,:);
```

```
    ndx=find(in~=-1);           % acha as entradas que disparam
    tam=size(ndx,2);
    t=0;                        % comeca em t=0
    neu=0;
%    h=figure; hold on;
    while neu==0 & t<=tmax
        out=zeros(1,out_neu);
        for j=1:out_neu         % para cada saida
            for i=1:tam
                if wx(ndx(i),j)==0 | cx(ndx(i),j)==0
                    sai=0;
                else
                    dt=(t-in(ndx(i))-cx(ndx(i),j))/3;
                    sai=wx(ndx(i),j)*dt*exp(1-dt);
                    if sai<0
                       sai=0;
                    end
                end
                out(j)=out(j)+sai;
            end
%            plot(t,out(j),t,teta);
        end
        if max(out)>=teta
            neu=find(out==max(out));
            inst=t;
        end
        t=t+rho;
    end
%    pause; close(h);
    if size(neu,2)>1    % se deu empate...
        neu=0;          % classe = 0
    end
    class(n)=neu;                          % guarda a classe
end
%-------------------------------------------------------------------------
```

```
% Plota resultado do teste
ptos={'.b' '*k' 'or' '+b' 'xk' 'sr' 'db'...
      '.k' '*r' 'ob' '+k' 'xr' 'sb' 'dk'...
      '.r' '*b' 'ok' '+r' 'xb' 'sk' 'dr'};
figure; hold on;
for i=1:out_neu
    a=find(class==i);
    for n=1:size(a,2)
        plot(aux1(a(n),1),aux1(a(n),2),char(ptos(i)));
    end
end
a=find(class==0);
for n=1:size(a,2)
    plot(aux1(a(n),1),aux1(a(n),2),char(ptos(out_neu+1)));
end
```

A.3.3 Using equation (5.9)

```
% cl_spk_5b - teste de classificacao com SNN
%
% Ultima alteracao 18/08/2005 - RCBerredo
%
% Usa 5 conjuntos com distribuicao gaussiana com media zero.
% Cada conjunto tem 50 pontos.
%
% Classes (saidas): 5
% Codificacao: Campos receptivos fixos
% Conj. de Campos Receptivos: 1
% Tipo de Sinapses: simples
% Treinamento: adapta pesos e ctes de tempo sao funcao do peso
%
%--------------------------------------------------------------------------
% Prepara Entrada
clear all;
aux1=dlmread('cj_spk5.txt');   % dados de entrada
%--------------------------------------------------------------------------
```

```
% Define Parametros Iniciais
out_neu=5;                      % qtde de neuronios na saida (classes)
rho=0.1;                        % resolucao de tempo
conj=125;                        % qtde de conjuntos de treinamento
%tau=3;                          % cte de tempo do EPSP
rf=[8 8];                       % qtde de neuronios por entrada
sigma=[1/(1.5*(rf(1)-2)) 1/(1.5*(rf(2)-2))]; % largura dos campos recep
f_cut=0.9;
maxd=10;                        % intervalo de codificacao
%-------------------------------------------------------------------------
% Define Parametros de Aprendizado
tau_max=9.8; tau_min=0.1;        % ctes de tempo maxima e minima
w_max=6.98; w_min=0;              % pesos maximo e minimo
max_epoch=10;                   % no. maximo de epocas
t_max=30;                       % tempo maximo de treinamento
eta_w=0.035;                     % taxa de aprendizado p/ pesos
beta=0.26;                       % cte para janela de aprendizado de pesos
nu=3;                         % vizinhanca
dx=2.8;                         % deslocamento de W(t) p/ esquerda (+) ou
%-------------------------------------------------------------------------
% Calcula Parametros
kapa=1-(nu^2)/(2*log(beta/(beta+1))); % cte para janela de aprendizado
in_neu=sum(rf);                       % qtde de neuronios na entrada
teta=12;                              % valor do limiar
%-------------------------------------------------------------------------
% Inicializa Pesos e Ctes de Tempo
w=w_min+rand(in_neu,out_neu).*(w_max-w_min);
tau=tau_min+rand(in_neu,out_neu).*(tau_max-tau_min);
%tau=tau_max.*(0.2452.*exp(-0.07922.*(w./w_max))...
%    +0.09115.*exp(2.149.*(w./w_max)));
%-------------------------------------------------------------------------
% inicia o treinamento
tic;
aus=kodieren_rf(aux1,rf,maxd,sigma,f_cut,rho,0,0);  % codifica entrada
ctrl_1=1;                                           % controle do loop
```

```
h=waitbar(ctrl_1/max_epoch,sprintf('Executando %i de %i iteracoes',...
  ctrl_1, max_epoch));
f_times=zeros(2,1);
while ctrl_1<=max_epoch
    for z=1:conj
        tr=aus(z,:);
        [neu, inst]=wer_spuckt(tr,t_max,w,tau,rho,teta,0);
        f_times=[f_times [neu;inst]];
        if neu~=0 & size(neu,2)==1
            for i=1:in_neu
                if tr(i)==-1
                    w(i,neu)=w(i,neu)-eta_w;
                    tau(i,neu)=tau_max*(0.2452*exp(-0.07922*(w(i,neu)/...
                            w_max))+0.09115*exp(2.149*(w(i,neu)/w_max)));
                else                        % ajusta pesos e ctes de tempo
                    delta_t=tr(i)-inst;      % calcula delta_t
                    w(i,neu)=w(i,neu)+eta_w.*((1+beta).*...
                             exp(-(((delta_t+dx).^2)./(2*kapa-2)))-beta);
                    tau(i,neu)=tau_max*(0.2452*exp(-0.07922*(w(i,neu)/...
                            w_max))+0.09115*exp(2.149*(w(i,neu)/w_max)));
                end
%                if w(i,neu)<=0
%                    tau(i,neu)=tau_min;
%                end
                % garante limites dos pesos e ctes de tempo
                if w(i,neu)>w_max
                    w(i,neu)=w_max;
                elseif w(i,neu)<w_min
                    w(i,neu)=w_min;
                end
%                if tau(i,neu)>tau_max
%                    tau(i,neu)=tau_max;
%                elseif tau(i,neu)<tau_min
%                    tau(i,neu)=tau_min;
%                end
```

```
            end
        end
    end
    ctrl_1=ctrl_1+1;                    % incrementa contador de epocas
    waitbar(ctrl_1/max_epoch,h,sprintf('Executando %i de %i iteracoes',...
    ctrl_1, max_epoch));
    teta=teta+(0.3*teta)/max_epoch;
    %show_w(w);
end
close(h);
toc;
%-------------------------------------------------------------------------
% testa
[conj,in_neu]=size(aus);
class=zeros(1,conj);            % cria tabela de classes
for i=1:conj
    tr=aus(i,:);                % conjunto de teste
    [neu,inst]=wer_spuckt(tr,t_max,w,tau,rho,teta,0);   % quem disparou?
    if size(neu,2)>1    % se deu empate...
        neu=0;          % classe = 0
    end
    class(i)=neu;                           % guarda a classe
end
% Plota resultado do teste
ptos={'.b' '*k' 'or' '+b' 'xk' 'sr' 'db'...
      '.k' '*r' 'ob' '+k' 'xr' 'sb' 'dk'...
      '.r' '*b' 'ok' '+r' 'xb' 'sk' 'dr'};
figure; hold on;
for i=1:out_neu
    a=find(class==i);
    for n=1:size(a,2)
        plot(aux1(a(n),1),aux1(a(n),2),char(ptos(i)));
    end
end
a=find(class==0);
```

```
for n=1:size(a,2)
    plot(aux1(a(n),1),aux1(a(n),2),char(ptos(out_neu+1)));
end
```

A.4 Auxiliary functions

```
function [neu, inst] = wer_schiesst(in, tmax, w, d, tau, rho, teta, plt)
%
% Funcao [neu, inst] = wer_schiesst(in, tmax, w, d, tau, rho, teta, plt)
%
% Retorna o neuronio que dispara e o instante de disparo.
% Entrada:
%           in = matriz linha com as entradas (firing times)
%         tmax = maximo intervalo de calculo. se nenhuma saida dispara
%                ate este instante, o calculo do EPSP termina
%            w = matriz m x n x k com os pesos sinapticos, onde m sao os
%                neuronios de entrada, n os de saida e k sub-sinapses.
%            d = matriz linha com os atrasos de cada sub-sinapse
%          tau = constante de tempo para calculo do EPSP
%          rho = resolucao de tempo
%         teta = valor do limiar
%          plt = se e 1 plota
%
% Saida:
%          neu = neuronios que disparam (Atencao: pode ser mais de um!)
%         inst = instante de disparo (se nehuma saida dispara inst = -1)
%
% Ultima alteracao em 19/07/2005 - RCBerredo
%
%-------------------------------------------------------------------------
[ein,aus,ssin]=size(w);        % ein = entradas (neuronios), aus = saidas
ndx=find(in~=-1);                % acha as entradas que disparam
tam=size(ndx,2);
if tam==0                        % se ninguem dispara, sai
    neu=0; inst=-1;
```

```
    return
end
t=0;                            % comeca em t=0
neu=0;
while neu==0 & t<=tmax
    out=zeros(1,aus);
    for j=1:aus
        for i=1:tam
            dt=t-in(ndx(i));
            for k=1:ssin
                dtt=(dt-d(k))/tau;
                sai=w(ndx(i),j,k)*dtt*exp(1-dtt);
                if sai<0
                    sai=0;
                end
                out(j)=out(j)+sai;
            end
        end
        if plt==1
            plota(aus,j,t,out(j),teta);
        end
    end
    if max(out)>=teta
        neu=find(out==max(out));
        inst=t;
    end
    t=t+rho;
end
if neu==0                   % se acabou limite de tempo
    inst=-1;
end
%-------------------------------------------------------------------------
function plota(p1, p2, px, py, teta)
    subplot(p1,1,p2); hold on;
    plot(px,py,'.',px,teta,'r.'); drawnow;
```

```
function [neu, inst] = wer_spuckt(in, tmax, w, tau, rho, teta, plt)
%
% Funcao [neu, inst] = wer_spucktt(in, tmax, w, tau, rho, teta, plt)
%
% Retorna o neuronio que dispara e o instante de disparo.
% Entrada:
%           in = matriz linha com as r entradas (firing times)
%         tmax = maximo intervalo de calculo. se nenhuma saida dispara
%                ate este instante, o calculo do EPSP termina
%            w = matriz m x n com os pesos sinapticos, onde m sao os
%                neuronios de entrada e n os de saida
%          tau = matriz m x n com as ctes de tempo de cada sinapse
%          rho = resolucao de tempo
%         teta = valor do limiar
%          plt = 1 - plota o EPSP
%                0 - nao plota
%
% Saida:
%          neu = neuronios que disparam (Atencao: pode ser mais de um!)
%         inst = instante de disparo (se nehuma saida dispara inst = -1)
%
% Ultima alteracao em 18/08/2005 - RCBerredo
%
%--------------------------------------------------------------------------
[ein,aus]=size(w);              % ein = entradas (neuronios), aus = saidas
ndx=find(in~=-1);               % acha entradas que disparam
tam=size(ndx,2);                % quantos disparam
if tam==0                       % se ninguem dispara, sai
    neu=0; inst=-1;
    return
end
t=0;                            % comeca em t=0
neu=0;
while neu==0 & t<=tmax
    out=zeros(1,aus);
```

```
    for j=1:aus
        for i=1:tam
            dt=(t-in(ndx(i)))/tau(ndx(i),j);
            sai=w(ndx(i),j)*dt*exp(1-dt);
                if sai<0
                    sai=0;
                end
            out(j)=out(j)+sai;
        end
        if plt==1
            plota(aus,j,t,out(j),teta);
        end
    end
    if max(out)>=teta
        neu=find(out==max(out));
        inst=t;
    end
    t=t+rho;
end
if neu==0                    % se acabou limite de tempo
    inst=-1;
end
%-------------------------------------------------------------------------
function plota(p1, p2, px, py, teta)
    subplot(p1,1,p2); hold on;
    plot(px,py,'.',px,teta,'r.'); drawnow;

function [c,u,saida,class] = Kmeans_var(xin, nc, maxepoca, tol)
%
%
% Esta funcao tem por objetivo determinar nc centros dado um conjunto
% de entrada xin (na forma de uma variavel) conforme descrito:
%
%                        xin = [x11  x12  ...  x1j;
%                               x21  x22  ...  x2j;
%                               ...  ...  ...  ...;
```

```
%                               xi1  xi2  ...  xij]
%
% Onde xij sao os j vetores para os quais se busca os centros, maxepoca
% e o numero maximo de iteracoes e tol e a variacao maxima (diferenca
% absoluta) entre duas posicoes consecutivas de cada nc centro.
% O metodo usado e o K-means. A saida tem os seguintes componentes:
%
% - c e uma matriz com os n centros. Cada linha representa um centro:
%
%                         c = [c11  c12  ...  c1j;
%                              c21  c22  ...  c2j;
%                              ...  ...  ...  ...;
%                              cn1  cn2  ...  cnj]
%
% - u e a matriz de pertinencia final, onde cada columa corresponde a um
% centro e cada linha a um dos vetores de entrada. Todas as colunas
% (centros) serao zero, exceto para o centro ao qual pertencer o vetor,
% sendo esta coluna igual a um.
%                              c1    c2  ...  cn
%                               |    |    |    |
%                         u = [u11  u12  ...  u1j; -->  xin1
%                              u21  u22  ...  u2j; -->  xin2
%                              ...  ...  ...  ...; -->  ...
%                              un1  un2  ...  unj] -->  xinn
%
% - saida e a matriz com as entradas (xin) normalizadas. Se nao for usada
% a normalizacao, saida e igual a xin.
%
% - class e uma matriz de uma coluna, com cada linha correspondendo a uma
% linha de "xin" ou "saida", indicando o centro ao qual pertence o vetor.
%
% Ultima alteracao em 28/12/2002 - RCBerredo
%
%
% Inicializar Variaveis
```

```
dJ=tol+1;                        % inicializa dJ com valor maior que tol
epoca=0;
lin=size(xin,1);               % quantidade de vetores (linhas)
col=size(xin,2);               % quantidade de colunas
%-------------------------------------------------------------------------
%aux=sqrt(sum(xin.^2,2));       % normaliza entrada
%for n=1:lin
%    input(n,:)=xin(n,:)./aux(n);
%end
%-------------------------------------------------------------------------
input=xin;                      % nao normaliza a entrada
%-------------------------------------------------------------------------
ndx=round(rand(1,nc).*lin);      % pega nc vetores aleatorios
% No futuro, testar para ver se tem vetores iguais...
c=zeros(nc,col);                   % inicializa a matriz de centros
for n=1:nc                         % monta matriz inicial de centros
c(n,:)=input(ndx(n),:);
end
while (dJ>tol & epoca<maxepoca)  % loop principal
    u=zeros(lin,nc);        % monta a matriz de pertinencia com zeros
    for vetor=1:lin
    for centro=1:nc   % calcula dist. entre cada vetor e os centros
    dist(centro)=sum((input(vetor,:)-c(centro,:)).^2);
    end
    um=find(dist==min(dist));  % acha a menor distancia e
    u(vetor,um)=1;             % poe 1 na matriz de pertinencia
    end
    for centro=1:nc                % para cada centro:
     unos=find(u(:,centro)==1); % acha vetores da classe nc
     soma1=0;                   % zera somas
    soma2=zeros(1,col);
     for membro=1:size(unos,1)  % soma distancias da classe nc
    soma1=soma1+sum(((input(unos(membro),:))-c(centro,:)).^2);
    soma2=soma2+(input(unos(membro),:));
        end
```

```
        dist1(centro)=soma1;
        c(centro,:)=soma2(1,:)./size(unos,1);  % calcula novo centro classe nc
    end
    epoca=epoca+1;
    J(epoca)=sum(dist1);            % calcula funcao de custo
    if epoca==1                     % ve se estabiliza
        dJ=J(epoca);
    else
        dJ=abs(J(epoca-1)-J(epoca));
    end
end
%-------------------------------------------------------------------------
%
% prepara saida
saida=input;
% constroi matriz de classes e centros
for k=1:lin
    class(k,1)=find(u(k,:)~=0);
end
x=1:1:epoca;
plot(x,J);
Title(sprintf('K-Means - Variacao = %0.5g',dJ),'FontSize',14);
xlabel('Epocas','FontSize',14);
ylabel('Soma das Distancias','FontSize',14);
%-------------------------------------------------------------------------
```

Appendix B

Sites on the Internet

The purpose of this appendix is to list a few links to home pages related with the present work and from where some of the material used here was obtained. These sites will certainly be of some use to those interested on this subject.

B.1 Softwares

http://www.genesis-sim.org/GENESIS/: GENESIS (short for GEneral NEural SImulation System) is a general purpose simulation platform that was developed to support the simulation of neural systems ranging from subcellular components and biochemical reactions to complex models of single neurons, simulations of large networks, and systems-level models. GENESIS has provided the basis for laboratory courses in neural simulation at Caltech, the Marine Biological Laboratory, the Crete, Trieste, Bangalore, and Obidos short courses in Computational Neuroscience, and at least 49 universities of which we are aware. Most current GENESIS applications involve realistic simulations of biological neural systems. Although the software can also model more abstract networks, other simulators are more suitable for backpropagation and similar connectionist modeling.

http://www.enorg.org/: This site of the Catacomb neural simulation software (Components And Tools for Acessible COmputer Modeling in Biology). Catacomb is a software system for building and studying models of biological systems on scales ranging from single channels to behaving animals. It incorporates graphical tools for creating model descriptions, and a

library of components for computing their behavior.

http://www.neuron.yale.edu/neuron/install/install.html: NEURON is a simulation environment for developing and exercising models of neurons and networks of neurons. It is particularly well-suited to problems where cable properties of cells play an important role, possibly including extracellular potential close to the membrane, and where cell membrane properties are complex, involving many ion-specific channels, ion accumulation, and second messengers.

http://www-ra.informatik.uni-tuebingen.de/SNNS/: SNNS (Stuttgart Neural Network Simulator) is a software simulator for neural networks on Unix workstations developed at the Institute for Parallel and Distributed High Performance Systems (IPVR) at the University of Stuttgart. The goal of the SNNS project is to create an efficient and flexible simulation environment for research on and application of neural nets.

http://www.math.pitt.edu/ bard/xpp/xpp.html: Home page of G. Bard Ermentrout, professor of Computational Biology at the University of Pittsburgh, containing useful mathematics packages for exploring phase spaces and dynamical systems, including XPP.

http://minduploading.org/oss/rak-ncm.html: This site contains various programs for neural circuitry modeling, mainly octave scripts and C++ programs.

http://www.cs.cmu.edu/ dst/HHsim: This is a simple educational software designed specifically for graduate or undergraduate neurophysiology courses. The user interface can be mastered in a couple of minutes and provides many ways for the student to experiment.

B.2 Other Sites

http://neurobranches.chez.tiscali.fr/neurophy/neurophy.html: Interesting independent site (in french) about neurophysiology, althought not very complete. Useful as a summary, rather than detailed information.

http://www.biologymad.com/NervousSystem/NervousSystem.htm: Another independent site about neurophysiology, with animated tutorials and dtailed descriptions of the main componentes of the nerve system.

http://www.siumed.edu/∼dking2/ssb/neuron.htm: Site made by Martin Ryder, professor of the University of Colorado at Denver. Here one can find information and writings by and about leading thinkers in cognitive science, and critics and observers of the philosophy of mind.

http://www.neuroinf.de/: The Neuroinformatics Portal Pilot project, funded through a grant from the German Ministry for Science and Education (BMBF), is part of a larger effort to promote the exchange of neuroscience data, data-analysis tools, and modeling software. It is now only a basic infrastructure needed for an optimal utilization of the many available resources at a later point in time. This Portal Pilot is already on-line to allow interested users to explore the possibilities of this prototype and thus start the feedback process needed to make this Portal a community-driven resource.

http://www.siumed.edu/ dking2/ssb/neuron.htm: Site made by avid King, professor of the Southern Illinois School of Medicine. This site is a complete and well organized guide to nerophysiology and contains very interesting information on this subject.

Bibliography

[AA99] Osamu Araki and Kazuyuki Aihara. Dual coding in network of spiking neurons: Aperiodic spikes and stable firing rates. In *IJCNN '99. International Joint Conference on Neural Networks*, volume 1, pages 514–518, Jul 1999.

[And04] Peter Andras. A model for emergent chaotic order in small neural networks. Technical Report CS-TR-860, School of Computer Science, University of New Castle upon Tyne, Sep 2004.

[ASY00] Kathleen T. Alligood, Tim D. Sauer, and James A. Yorke. *Chaos – An Introduction to Dynamical Systems*. Springer Verlag, New York, 3rd edition, 2000.

[AT02] Kazuyuki Aihara and Isao Tokuda. Possible neural coding with interevent intervals of synchronous firing. *Physical Review E*, 66:26212–1–5, Aug 2002.

[BB03] James M. Bower and David Beeman. *The Book of Genesis – Exploring Realistic Neural Models with the GEneral NEural SImulation System*. Internet Version, 2nd edition, 2003. http://www.genesis-sim.org/GENESIS.

[BC84] J P Baltanás and J M Casado. Noise-indiced resonances in the hindmarsh-rose neuronal model. *Physical Review E*, 65:41915–1–6, 1984.

[BKP02] Sander M. Bohte, Joost N. Kok, and Han La Poutre. Error back-propagation in temporally encoded networks of spiking neurons. *Neuro Computing*, 48:17–37, 2002.

[BLdC00] A P Braga, T B Ludemir, and André C P L F de Carvalho. *Artificial Neural Networks – Theory and Applications (in portuguese)*. LTC Editora, Rio de Janeiro, 1st edition, 2000.

[Boh03] Sander Marcel Bohte. *Spiking Neural Networks.* PhD thesis, Universiteit Leiden, Mar 2003.

[Boh04] Sander Marcel Bohte. The evidence of information processing with precise spike-times: A survey. *Natural Computing*, 3:195–206, 2004.

[BPK02] Sander M. Bohte, Han La Poutré, and Joost N. Kok. Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks. *IEEE Transactions on Neural Networks*, 13(2), Mar 2002.

[BSS93] N. Barkai, H. S. Seung, and H. Sompolinsky. Scaling laws in learning of classification tasks. *Physical Review Letters*, 70(20):3167–3170, May 1993.

[CBC02] Chris Christodoulou, Guido Bugmann, and Trevor G. Clarkson. A spiking neuron model: Applications and learning. *Neural Networks*, 15(7):891–908, Sep 2002.

[Coo03] S. Coombes. Dynamics of synaptically coupled integrate-and-fire-or-burst neurons. *Physical Review E*, 67:41910–1–10, Apr 2003.

[DA01] Peter Dayan and L. F. Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems.* The MIT Press, Cambridge, 1st edition, 2001.

[dKvdV01] Marc de Kamps and Frank van der Velde. From artificial neural networks to spiking neuron populations and back again. *Neural Networks*, 14(6–7):941–953, Jul 2001.

[dQ02] Murilo Saraiva de Queiros. Reinforcement learning in spiking neurons based on the spike response model. Master's thesis, Universidade Federal de Minas Gerais, Aug 2002.

[EPE+99] Christian W Eurich, Klaus Pawelzik, Udo Ernst, Jack D Cowan, and John G Milton. Dynamics of self-organized delay adaptation. *Physical Review Letters*, 82(7):1594–1597, Feb 1999.

[EPE+00] Christian W Eurich, Klaus Pawelzik, Udo Ernst, Andreas Thiel, Jack D Cowan, and John G Milton. Delay adaptation in the nervous system. *Neurocomputing*, 32–33:741–748, Set 2000.

[Fit61] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, 1:445–466, 1961.

[Ger99] Wulfram Gerstner. Rapid signal transmission by populations of spiking neurons. In *ICANN 99. Ninth International Conference on Artificial Neural Networks*, volume 1, pages 7–12, Sep 1999.

[Ger01] Wulfram Gerstner. What´s different with spiking neurons? In Henk Mastebroek and Hans Vos, editors, *Plausible Neural Networks for Biological Modelling*, pages 23–48. Kluwer Academic Publishers, 2001.

[GK02a] Wulfram Gerstner and Werner M. Kistler. Mathematical formulations of Hebbian learning. *Biological Cybernetics*, 87:404–415, 2002.

[GK02b] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models*. The Cambridge University Press, Cambridge, 1st edition, 2002.

[GLR80] R. Guttman, S. Lewis, and J. Rinzel. Control of repetitive firing in squid axon membrane as a model for a neurooscillator. *Journal of Physiology*, 305:377–395, 1980.

[Gra93] Paul Gray. What is love? *Time Magazine*, 141(7), 1993.

[Has00] Hideo Hasegawa. Responses of a Hodgkin-Huxley neuron to various types of spike-train inputs. *Physical Review E*, 61(1):718–726, Jan 2000.

[Hay99] Simon Haykin. *Neural Networks: a compreensive foundation*. Prentice Hall, 2nd edition, 1999.

[Heb49] D. O. Hebb. *The Organization of Behaviour*. John Wiley & Sons, New York, 1949.

[Häf00] Philipp D. Häfliger. *A Spike Based Learning Rule and its Implementation in Analog Hardware*. PhD thesis, Swiss Federal Institute of Technology, 2000.

[HH52] A L Hodgkin and A F Huxley. A quantitative description of ion currents and its applications to conduction and excitation in nerve membranes. *Journal of Physiology*, 117:500–544, 1952.

[HI01] Frank C. Hoppensteadt and Eugene M. Izhikevich. Canonical neural models. In M.A. Arbib, editor, *Brain Theory and Neural Networks*. The MIT Press, 2nd. edition, 2001.

[HL03] Duane Hanselman and Bruce Littlefield. *Mastering MATLAB 6: A Compreensive Tutorial and Reference (in portuguese)*. Prentice Hall, 2003.

[HMR02] Gerardina Hernández, Paul Munro, and Jonathan Rubin. Mapping from the spike domain to the rate-based domain. In *ICONIP '02. 9th International Conference on Neural Information Processing*, volume 4, pages 1791–1795, Nov 2002.

[Hop88] John J. Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, pages 3–10, Sep 1988.

[Hop95] John J. Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376:33–36, 1995.

[HR82] J L Hindmarsh and R M Rose. A model of the nerve impulse using two first-order differential equations. *Nature*, 296:162–164, 1982.

[HR84] J L Hindmarsh and R M Rose. A model of neuronal bursting using three coupled first order differential equations. *Proc. R. Soc. Lond. Biol.*, 221:87–102, 1984.

[IGE04] Eugene M Izhikevich, Joseph A Gally, and Gerald M Edelman. Spike-timing dynamic of neuronal groups. *Cerebral Cortex*, 14:933–944, Aug 2004.

[IMI99] S. Inawashiro, S. Miyake, and M. Ito. Spiking neuron models for regular-spiking, intrinsically bursting, and fast-spiking neurons. In *ICONIP '99. 6th International Conference on Neural Information Processing*, volume 1, pages 32–36, Nov 1999.

[Izh01] Eugene M. Izhkevich. Resonate-and-fire neurons. *Neural Networks*, 14(6–7):883–894, Jul 2001.

[Izh03] Eugene M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, Nov 2003.

[Izh04] Eugene M. Izhikevich. Which model to use for cortical spiking neurons. *IEEE Transactions on Neural Networks*, 15(5):1063–1070, Sep 2004.

[Jin04] Dezhe Z. Jin. Spiking neural network for recognizing spatiotemporal sequences of spikes. *Physical Review E*, 69:21905–1–13, Feb 2004.

[KA05] Takashi Kohno and Kazuyuki Aihara. A MOSFET-based model of a class 2 nerve membrane. *IEEE Transactions on Neural Networks*, 16(3):754–773, May 2005.

[KGvH99a] Richard Kempter, Wulfram Gerstner, and J. Leo van Hemmen. Hebbian learning and spiking neuron. *Physical Review E*, 4(59):4498–4514, Apr 1999.

[KGvH99b] Richard Kempter, Wulfram Gerstner, and J. Leo van Hemmen. Spike-based compared to rate-based Hebbian learning. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Proc. of NIPS 1999, Advances in Neural Information Processing Systems*, volume 11, pages 125–131, Cambridge, 1999. The MIT Press.

[KIKY03] Susumo Kamoi, Rie Iwai, Hiroshi Kinjo, and Tetsuhiko Yamamoto. Pulse pattern training of spiking neural networks using improved genetic algorithm. In *2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 2, pages 977–981, Jul 2003.

[Kin91] Chris C. King. Fractal and chaotic dynamics in nervous systems. *Progress in Neurobiology*, 36(4):270–308, 1991.

[Koc99] Christof Koch. *Biophysics of Computation: Information Processing in Single Neurons*. Oxford University Press, New York, 1st edition, 1999.

[KSJ00] Eric R Kandel, James H Schwartz, and Thomas M Jessell. *Principles of Neural Sciences*. McGraw-Hill, 4th edition, 2000.

[LKvH02] Christian Leibold, Richard Kempter, and J. Leo van Hemmen. How spiking neurons give rise to a temporal-feature map: From synaptic plasticity to axonal selection. *Physical Review E*, 65:51915–1–20, May 2002.

[LMB00] Nicolas Langlois, Pierre Miché, and Abdelaziz Bensrhair. Analogue circuits of a learning spiking neuron model. In *IJCNN 2000. International Joint Conference on Neural Networks*, volume 4, pages 485–489, Jul 2000.

[LvH01] Christian Leibold and J. Leo van Hemmen. Temporal receptive fields, spikes, and Hebbian delay selection. *Neural Networks*, 14(6–7):805–813, Jul 2001.

[Maa97] Wofgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[MB98] Wolfgang Maass and Christopher M. Bishop, editors. *Pulsed Neural Networks*. The MIT Press, Cambridge, 1st edition, 1998.

[MB00] R. Malaka and S. Buck. Solving nonlinear optimization problems using networks of spiking neurons. In *IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, volume 6, pages 486–491, Jul 2000.

[MG02] Maurizio Mattia and Paolo Del Giudice. Population dynamics of interacting spiking neurons. *Physical Review E*, 66:51917–1–19, Nov 2002.

[ML81] C Morris and H Lecar. Voltage oscillations in the barnacle giant muscle fiber. *Biophysical Journal*, 35:193–213, 1981.

[MP43] W S McCulloch and W Pitts. A logical calculus and the ideas immanent in the nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MP88] Marvin L Minsky and Seymour A Papert. *Perceptrons*. The MIT Press, Cambridge, 1969 (Enlarged edition, 1988).

[MZ99] W. Maass and A. M. Zador. Dynamic stochastic synapses as computational units. *Neural Computation*, 11(4):903–917, 1999.

[Nat96] Thomas Natschläger. Spiking neurons networks: The third generation (in german). In *Jenseits von Kunst*. Passagen Verlag, 1996.

[NAY62] J Nagumo, S Arimoto, and S Yoshizawa. An active pulse transmission line simulating nerve axon. *Proc. Institute of Radio Engineers (IRE)*, 50:2061–2070, 1962.

[NMZ01] T. Natschläger, W. Maass, and A. Zador. Efficient temporal processing with biologically realistic dynamic synapses. *Network: Computation in Neural Systems*, 12:75–87, 2001.

[NR66] K. I. Naka and W. A. Rushton. S-potentials from colour units in the retina of fish. *Journal of Physiology*, 185:584–599, 1966.

[NSD02] Christian Näger, Jan Storck, and Gustavo Deco. Speech recognition with spiking neurons and dynamic synapses: a model motivated by the human auditory pathway. *Neurocomputing*, 44–46:937–942, Jun 2002.

[PAF+04] Dale Purves, George J. Augustine, David Fitzpatick, William C. Hall, Anthony-Samuel LaMantia, James O. McNamara, and S. Mark Williams, editors. *Neuroscience*. Sinauer Associates, Inc., 3rd edition, 2004.

[PP95] Lucas C. Parra and Barak A. Pearlmutter. Maximal information transfer in a spiking neuron. *Society for Neuroscience Abstracts*, 21(480.7), 1995.

[RH89] R. M. Rose and J. L. Hindmarsh. The assembly of ionic currents in a thalamic neuron I. the tridimensional model. *Proceedings of the Royal Society of London, Series (B): Biological Sciences*, 237:267–288, 1989.

[RH05] Richard Reeve and John Hallam. An analysis of neural models for walking control. *IEEE Transactions on Neural Networks*, 16(3):733–742, May 2005.

[Ros58] F Rosenblatt. The Perctptron: A probabilistic model for information storage and organization in the brain. *Phsycological Review*, 65:386–408, 1958.

[Ruf98] Berthold Ruf. *Computing and Learning with Spiking Neurons – Theory and Simulations*. PhD thesis, Technische Univesität Graz, Austria, May 1998.

[Rul02] Nikolai F. Rulkov. Modeling of spiking-bursting neural behavior using two-dimensional map. *Physical Review E*, 65:41922–1–9, Apr 2002.

[RWdRvSB97] Fred Rieke, David Warland, Rob de Ruyter van Steveninck, and William Bialek. *Spikes – Exploring the Neural Code*. The MIT Press, Cambridge, 1st edition, 1997.

[SARC03] Tanya Sienko, Andrew Adamatzky, Nicholas G. Rambidi, and Michael Conrad, editors. *Molecular Computing*. MIT Press, Cambridge, 1st edition, 2003.

[SC99] Dorel M. Sala and Krzysztof J. Cios. Solving graph algorithms with networks of spiking neurons. *IEEE Transactions on Neural Networks*, 10(4):953–957, Jul 1999.

[SHY00] R. Sarpeshkar, R. Herrera, and H. Yang. A current-mode spike-based overrange-subrange analog-to-digital converter. In *ISCAS 2000. IEEE International Symposium on Circuits and Systems*, volume 4, pages 397–400, May 2000.

[SKA01] Yuichi Sakumura, Norio Konno, and Kazuyuki Aihara. Approximating markov chain model of the Hodgkin-Huxley neuron. *Trans. of the Materials Research Society of Japan*, 26(1):457–460, 2001.

[SKdRvSB98] S. P. Strong, Roland Koberle, Rob R. de Ruyter van Steveninck, and William Bialek. Entropy and information in neural spike trains. *Physical Review Letters*, 80(1):197–200, Jan 1998.

[Sni96] H P Snippe. Parameter extraction from population codes: A critical assesment. *Neural Computation*, 8(4):511–529, 1996.

[SS02] Yasuomi D. Sato and Masatoshi Shiino. Spiking neuron models with excitatory or inhibitory synaptic couplings and synchronization phenomena. *Physical Review E*, 66:41903–1–13, Oct 2002.

[SSW+02] Martin Schäfer, Tim Schönhauer, Carsten Wolf, Georg Hartmann, Heinrich Klar, and Ulrich Rückert. Simulation of spiking neural networks – architectures and implementations. *Neurocomputing*, 48(1–4):647–679, Oct 2002.

[SW99] Volker Steuber and David J. Willshaw. Adaptive leaky integrator models of cerebellar purkinje cells can learn the clustering of temporal patterns. *Neurocomputing*, 26:271–276, 1999.

[SZK99] Jeong-Woo Sohn, Byoung-Tak Zhang, and Bong-Kiun Kaang. Temporal pattern recognition using a spiking neural network with delays. In *IJCNN '99. International Joint Conference on Neural Networks*, volume 4, pages 2590–2593, Jul 1999.

[TH86] David W. Tank and John J. Hopfield. Simple "Neural" optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit. *IEEE Transactions on Circuits and Systems*, cas-33(5):533–541, May 1986.

[TNS02] Yusuke Takahashi, Hidehiro Nakano, and Toshimichi Saito. A four-dimensional hyperchaotic spiking neuron. In *ICONIP '02. 9th International Conference on Neural Information Processing*, volume 1, pages 358–362, Nov 2002.

[Tou93] Anastasia Toufexis. The right chemistry. *Time Magazine*, 141(7), 1993.

[Vol04] Markus Volkmer. A pulsed neural network model of spectro-temporal receptive fields and population coding in auditory cortex. *Natural Computing*, 3:177–93, 2004.

[vVS96] C. van Vreeswijk and H. Sompolinsky. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, 274:1724–1726, Oct 1996.

[WF02] Chih-Hsiu Wei and Chin-Shyurng Fahn. The multisynapse neural network and its application to fuzzy clustering. *IEEE Transactions on Neural Networks*, 13(3):600–618, May 2002.

[Wil99] Hugh R. Wilson. *Spikes Decisions and Actions – Dynamical Foundations of Neuroscience*. The Oxford University Press, Oxford, 1st edition, 1999.

[WZR98] Ervin Wolf, Fei-Yue Zhao, and Alan Roberts. Non-linear summation of excitatory sinaptic inputs to small neurones: a case study in spinal motoneurones of the young *Xenopus* tadpole. *Journal of Physiology*, 511.3:871–886, 1998.

[XE01] Jianguo Xin and Mark J. Embrechts. Supervised learning with spiking neural networks. In *IJCNN 2001. International Joint Conference on Neural Networks*, volume 3, pages 1772–1777, Jul 2001.

[yC03] Santiago Ramon y Cajal. *Texture of the Nervous System of Man and the Vertebrates*. Springer Verlag, New York, 2003. An annotated and edited translation of the original Spanish text with the additions of the French version by Pedro Pasik and Tauba Pasik (editors) – 3 volume set.

[YS98] Masahiko Yoshioka and Masatoshi Shiino. Associative memory based on synchronized firing of spiking neurons with time-delayed interactions. *Physical Review E*, 58(3):3628–3639, Sep 1998.

[YS99] Masahito Yoshioka and Masatoshi Shiino. Pattern coding based on firing times in a network of spiking neurons. In *IJCNN '99. International Joint Conference on Neural Networks*, volume 1, pages 444–449, Jul 1999.