Doctoral Thesis

Computing and Learning with Spiking Neurons – Theory and Simulations

Berthold Ruf

Institute for Theoretical Computer Science
Technische Universität Graz, Austria

May 5, 1998

Acknowledgments

This thesis has benefited from several suggestions and ideas from numerous people. I am especially thankful to:

- Wolfgang Maass, my supervisor, who pointed me to the exciting subject of spiking neurons and who helped me in many discussions refining my ideas. His work provided the main framework of this thesis. Furthermore, Chapter 4 is the result of some joint work with him.
- Helmut Pockberger for his friendly willingness to be the second referee of this thesis and for discussing several issues of this thesis with me.
- Michael Schmitt, with whom I worked on the topics described in Chapter 6 and 7.
- Thomas Natschläger, with whom I worked on the topics presented in Chapter 8.
- Peter Auer and Oswin Aichholzer, who discussed with me several issues of Chapter 8.
- various fruitful discussions with other researchers, especially with Walter Senn, Henry Markram, Pekka Orponen, Wulfram Gerstner, Ulrich Roth and Wolf Singer.

Abstract

There is strong evidence that biological neurons encode through their firing information not only in the firing rate but also in the timing of single spikes. This thesis explores various ways for computing and learning with networks of simplified spiking neurons (essentially of the leaky integrate-and-fire type) using temporal coding. We present both supervised and unsupervised learning rules, which are Hebbian in the sense that the strength of a synapse is modified if a pre- and a postsynaptic spike arrive at the synapse within a certain learning window. Recent neurobiological findings have confirmed such a dependency and have shown that the sign and strength of the change depends on the timing of the two spikes. On the basis of these principles we show how methods originally designed for artificial neural networks like competitive learning, self-organizing behavior and radial basis functions can be realized within this context. We also address the question whether these results still hold for biologically more realistic neurons and show by computer simulations for example that such neurons can compute linear functions in a very natural way.

Inhaltsangabe

Es gilt mittlerweile als praktisch gesichert, daß biologische Neuronen Informationen nicht nur über ihre Feuerrate sondern auch durch die Zeitpunkte, an denen sie einzelne Spikes aussenden, austauschen. Diese Arbeit untersucht unterschiedliche Möglichkeiten, wie Lernen und Rechnen mit Netzen von feuernden (spiking) Neuronen realisiert werden kann. Die dabei betrachteten Neuronen sind im wesentlichen vom sogenannten "leaky integrate-and-fire" Typ. Es werden Regeln zum überwachten und unüberwachten Lernen angegeben, welche als Hebb-Lernregeln bezeichnet werden können in dem Sinne, daß die Stärke einer Synapse modifiziert wird, falls ein prä- und ein postsynaptischer Spike an dieser Synapse innerhalb eines gewissen Lernfensters eintreffen. Solch eine Abhängigkeit wurde vor kurzem durch neurobiologische Experimente bestätigt. Es hat sich gezeigt, daß das Vorzeichen und die Stärke dieser

Änderung von den Zeitpunkten der beiden Spikes abhängt. Auf dieser Basis wird gezeigt, wie Verfahren, die ursprünglich für künstliche neuronale Netze entworfen wurden wie selbstorganisierende Karten und radiale Funktionen (radial basis functions), in diesem Kontext realisiert werden können. Außerdem wird untersucht, inwiefern solche Berechnungen auch mit biologisch realistischeren Neuronen möglich sind. Mit Hilfe von Computersimulationen wird beispielsweise gezeigt, daß solche Neuronen lineare Funktionen auf eine sehr einfache und natürliche Art berechnen können.

Contents

List of Figures

List of Main Variables

Name	Meaning	Described on page
$d_{u,v}$	Delay between neuron u and v	30
$d_{u,v}^{pre}$	Presynaptic part of $d_{u,v}$	30
$d_{u,v}^{post}$	Postynaptic part of $d_{u,v}$	30
E_{rest}	Resting membrane potential	17
m_{kj}	Neighborhood value between neuron v_j and v_k	
$P_v(t)$	Membrane potential of neuron v at time t	29
$\mathbf{R}^+$	Set of all positive reals including 0	
$\mathbf{s}$	Input vector	62
$\mathbf{w}$	Weight vector	
w_{ij}	Weight from neuron u_i to v_j	
$\tilde{w}_{ij}$	Lateral weight from neuron v_i to v_j	62
$\mathbf{x}$	Input vector in competitive coding	70
T_{in}	Time constant denoting the end of the coding interval	48
T_{out}	Time constant denoting the latest time	
	the output neuron may fire	50
Δ	Length of the initial segment of an PSP	
η	Learning rate	54, 59, 63, 73
$\varepsilon_{u,v}(t)$	Response function from neuron u to v at time t	29
κ	Factor for weight scaling	89
λ	Sum of all weights of a neuron v_j: $\lambda = \sum_{i=0}^{m} w_{ij}$	50
$\sigma_{u,v}$	Length of non-zero segment of $\varepsilon_{u,v}$	35
$\tau_{v,ref}$	Length of absolute refractory period	29
$\tau_{v,end}$	Length of (absolute + relative) refractory period	29
$\Theta_v(t)$	Threshold function for neuron v at time t	29
γ	Length of the coding interval	48
$\tilde{\gamma}$	Length of the coding interval for RBF neurons	70

1 Introduction

Artificial neural networks (ANNs), whose functioning is inspired by some fundamental principles of real biological neural networks, have proven to be a powerful computing paradigm. Real biological neurons communicate through short pulses, called “spikes”, which are omitted at varying rates. It is this firing rate which has been considered for a long time as the relevant information being exchanged between neurons. Also the (analog) inputs for an artificial neuron are usually interpreted as firing rates.

There exists strong evidence that it is inevitable for understanding the neural code, with which biological neurons communicate, to investigate the temporal structure of the spike trains. A neuron can emit spikes very irregularly; by averaging over a certain time window one would ignore a lot of information provided by the timing of the single spikes. In addition, it is obvious that especially in the context of fast information processing there is not always sufficient time to sample an average firing rate (see Chapter 2 for references).

Hence it seems very promising to extend the computational possibilities of ANNs by considering spiking neuron networks, which allow to take the timing of single spikes into account. The exploration of computational possibilities with SNNs may lead to new computing paradigm suitable for software realizations but also for fast hardware implementations, e.g. through pulsed VLSI. In addition, it may also help interpreting the neural code used by biological neurons.

So far there is not much known about possible computational mechanisms on the basis of the timing of single spikes. Some fundamental results have been provided recently by Maass: for example, he characterized the computational power of SNNs [Maass, 1996] and showed how the timing of spikes can be used to simulate sigmoidal gates with SNNs [Maass, 1997a].

This thesis addresses several question in this context: after summarizing some biological and computational foundations in Chapter 2 and 3, we discuss in Chapter 4 some aspects about the computational power of such SNNs. In [Maass, 1996] it was shown that SNNs can be computationally very powerful under certain assumptions. Here we ask the questions which of these assumptions are essential for maintaining such results and give rigorous proofs that the computational power of SNNs drops considerably if the neurons receive pulses (piecewise constant postsynaptic potentials)) as input. In Chapter 5 we summarize a construction provided in [Maass, 1997a] for computing a weighted sum in temporal coding with SNNs, which gives rise to simulating sigmoidal gates with SNNs.

There are still many fundamental questions about SNNs where not much is known about, but whose answers are crucial for considering SNNs as real extensions of ANNs, such as:

- If one includes certain features of biological neurons into SNNs, how does this influence the computational power?
- How can learning be realized with SNNs using temporal coding?
- How can the timing of spikes be used to store and analyze patterns or to achieve a self-organizing behavior?

This thesis suggests several learning mechanisms which obey the following philosophy: each connection to be modified by some learning rule is changed only due to locally available information which is provided by the timing of spikes produced by the two corresponding neurons. This strategy can be seen as a formulation of the Hebbian rule in the temporal domain and is motivated by recent neurobiological findings (see Section 2.4).

We discuss both supervised and unsupervised methods: in Chapter 6 we present a supervised learning rule for storing temporally encoded values into a weight of a synapse with the help of an error signal. We introduce a "monosynaptic" learning rule, where only one synapse at a time can be trained and then show how several synapses can be trained in parallel. For both methods proofs are provided.

Moreover, we show that such Hebbian learning in the temporal domain allows the adaptation of well-known types of ANNs with SNNs: in Chapter 7 we introduce an unsupervised learning rule realizing competitive learning with SNNs, where the competition and the determination of the winner neuron is implemented in a very fast, local and natural way by using lateral connections among the competing neurons. Then we present on the basis of this rule the first implementation of Kohonen's self-organizing map for spiking neurons using temporal coding. We will show that one of the main advantages of this coding scheme is that in contrast to traditional implementation of Kohonen's algorithm the computation of the neighborhood function can be realized fast and locally.

In Chapter 8 a fast way for learning radial basis functions with SNNs is introduced. In contrast to the previous sections the relevant information is stored here not in the weights but in the delays of connections between neurons. Learning of delays is realized by assuming multiple connections of various delays between pairs of neurons such that a shift of a delay can be achieved by modifying the corresponding weights. By taking the firing times of the corresponding output neurons into account, this approach can be extended to finding clusters in the input space, where the neurons express through their firing time to which cluster a certain input presented to the network belongs. This clustering is even in a dynamically changing environment possible, where during the learning process clusters may be added or removed. Furthermore, we will see that one can perform some kind of feature extraction within this model and learn temporal sequences, which can afterwards be recognized, even if they are distorted in various ways.

Our investigations are based on a rather simple model of a spiking neuron (the leaky integrate-and-fire neuron). The question arises whether biologically more realistic neurons, which fulfill only approximately some assumptions underlying our constructions, can perform the described computations. With the help of computer simulations we demonstrate in Chapter 9 that such neurons can compute linear functions in a very natural way: it turns out that for a wide range of parameters a neuron which receives one spike from each of its inputs within a certain time intervall fires at a time which depends in a linear fashion on the input vector encoded by these input spikes. Hence the main constructions presented in this work can still be realized by neurons which include some fundamental features of biological neurons. This chapter also shows how a simple way of pattern analysis can be realized with such neurons.

We would like to emphasize that this thesis is not about biology but about possibilities of computing with spiking neurons which are inspired by biology. We do not claim that any of the mechanisms described here do occur in exactly the same way in biological systems. However, a thorough understanding of such networks, which are rather simplified in comparison to real biological networks, may be helpful for detecting and understanding possible similar mechanisms in biological systems.

2 Biological Foundations

The elementary processing unit in nervous systems is the neuron. Neurons have an extremely complex structure and there exists a large variety of different types of neurons. This chapter summarizes some of the basic features of biological neurons where those aspects will be emphasized which will be of importance in the following.

2.1 The Basic Neuron

Figure 2.1 shows a typical biological neuron: its cell body, the *soma*, receives inputs through its *dendrites* and can produce as an output a *spike* (also called an action potential) which is propagated along its *axon* to other neurons. The axon may be considered as a long wire initiating at the *axon hillock* at the soma and branching into the *axonal arborization*. On a tip of such a branch the neuron may form a connection onto the dendrite (or sometimes onto the soma) of another neuron through a *synapse*.

Neurons function essentially on an electro-chemical basis: the interior of a neuron is separated from its surrounding by a membrane containing numerous ion channels causing a potential difference across the membrane, the *membrane potential*. An ion channel is highly specific (i.e. permeable) to one sort of ions, one distinguishes essentially between Na^+, K^+, Ca^{2+} and Cl^- channels. Usually the concentration of these ions differs between the interior and the surrounding of the cell, causing an osmotic pressure which tends to even out this difference. For example, the concentration of sodium ions is usually much higher in the extracellular fluid than in the interior of the cell. The concentration gradient resulting from the flow of sodium ions into the cell causes also a charge displacement opposing this concentration gradient. Eventually the potential will reach an equilibrium, the so-called *reversal potential*, where there is no net flux of sodium ions. If the neuron receives no input its membrane potential reaches the so-called *resting membrane potential* E_{rest}, which is usually between -40 and -90 mV depending on the type of neuron.

Each ion channels has a certain conductance. One has to distinguish between active and passive ion channels. Passive ion channels are permanently permeable to one sort of ions, thus having a constant conductance. In contrast to this the conductance of active ion channels may vary, depending on the current voltage or other factors.

If the membrane potential changes such that it exceeds at the axon hillock a certain

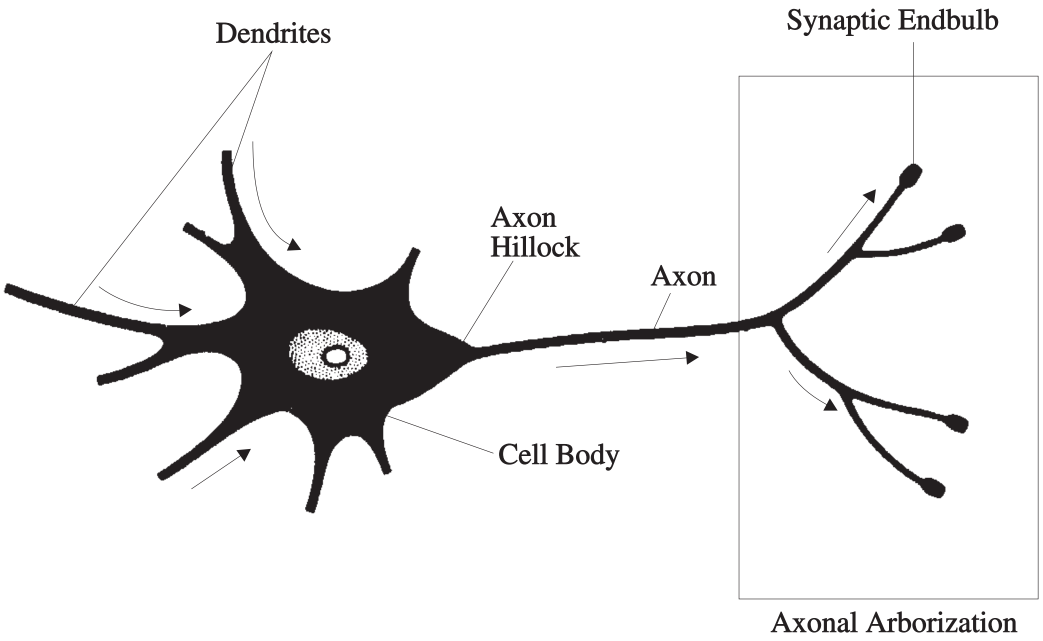

Figure 2.1: Schematic view of a neuron (after [Arbib, 1989], p. 52).

threshold (usually around - 30 mV), a spike is generated at the axon hillock, i.e. an undiminishing pulse which propagates down the axon. Immediately after the generation of an action potential the neuron enters a *refractory period* which can be divided into two parts: during the absolute refractory period the neuron is not able to generate a spike, i.e. the threshold is infinitely high. Afterwards the neuron enters a relative refractory period, during which the threshold approaches from above its normal value. Hence a stronger depolarization of the membrane potential is necessary during this period for the generation of a spike.

A spike which is propagated along the axon may finally branch to several synaptic endbulbs. A synapse then converts this presynaptic spike to a *postsynaptic potential* (PSP) which influences the membrane potential of that neuron. If the membrane potential is raised (thus increasing the probability of a spike generation), one speaks of an *excitatory postsynaptic potential* (EPSP), if it is lowered, making a spike less likely, one speaks of an *inhibitory postsynaptic potential* (IPSP). Figure 2.2 shows typical shapes of an EPSP and an IPSP. The height of EPSPs/IPSPs depends on the strength of the synapse and may vary with the time (see Section 2.4).

The synapses of a neuron may be placed on the dendritic tree or at the soma. Hence PSPs can interact in space and time. In the simplest case one assumes that the PSPs simply sum up linearly in the soma. This model is called an "integrate-and-fire neuron" (IFN). In most cases one considers an extension of the IFN model, where the membrane is assumed to be leaky due to ion channels, such that after a PSP the membrane potential approaches again E_{rest}. Such a neuron is called a "leaky integrate-and-fire neuron"

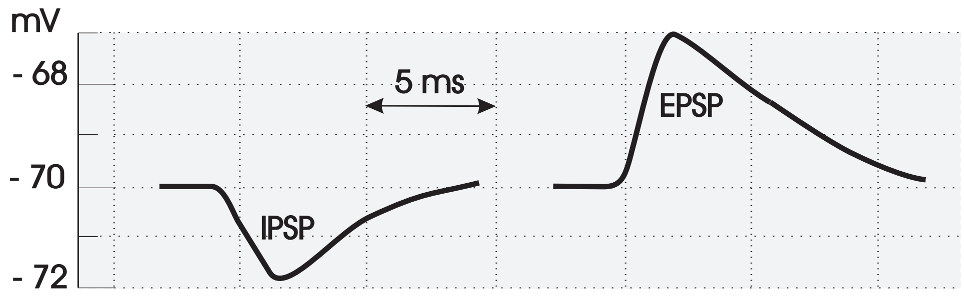

Figure 2.2: Typical shape of an EPSP and an IPSP (from [Churchland and Sejnowski, 1992], p. 43).

(LIFN) . A formal generalized description of this model will be given in Chapter 3. However, in real biological neurons, there are several nonlinear effects, e.g. the resulting potential at the soma of a neuron may be actually higher or lower than one might expect from the summation of the current values of the incoming PSPs. This topic will be discussed in more detail in the next section.

2.2 Non-linear Effects in Neurons

The LIFN model is a very simplistic model, which does not take into account various types of nonlinearities occurring in real neurons. Two of the main factors for these nonlinearities are the voltage-gated channels, described by the Hodgkin-Huxley model and the interaction of PSPs in the dendritic tree.

In the next chapters we will investigate various possibilities of computation with the LIFN model; in Chapter 9 we will then focus on the question, whether the simplifying assumptions of the LIFN model can be dropped. Here we give a short description of the main phenomena causing these non-linear effects.

2.2.1 The Hodgkin-Huxley Model

The closer the membrane potential gets to the threshold, the stronger is the non-linear influence of active, voltage-gated channels on the membrane potential. This influence was described for the first time in [Hodgkin and Huxley, 1952] for the squid giant axon. This landmark work is still the basis of our understanding of the initiation and propagation of action potentials.

In the squid giant axon, the membrane potential V_m is determined besides its passive ion channels, resulting in a voltage-independent leak conductance g_l, by active K^+ and Na^+ channels. These channels have voltage-dependent conductances g_K and g_{Na}. The concentration of sodium ions is in the intracellular axoplasm much smaller than in the extracellular fluid, the opposite holds for the potassium ions. The resulting reversal potentials are $E_l = -54.3\,mV$, $E_{Na} = 50\,mV$ and $E_K = -77\,mV$, the resting membrane potential is about $E_{rest} = -65\,mV$. The Hodgkin-Huxley model describes now the time dependencies of the active ion channels and the influence of their conductances on V_m. It can be best described through an equivalent circuit as shown in Figure 2.3. Each conductance is in series with a battery defining the reversal potentials. The circuit can be described now by

$$C\frac{dV_m}{dt} = g_l(E_l - V_m) + g_{Na}(E_{Na} - V_m) + g_K(E_K - V_m) \tag{2.1}$$

where C is the membrane capacitance. Each term $g_i(E_i - V_m)$ describes the ionic current resulting from the potential difference $E_i - V_m$. The conductance g_l is constant, whereas g_{Na} and g_K are time- and voltage-dependent:

$$\begin{aligned} g_{Na} &= G_{Na}m^3h \\ g_K &= G_K n^4 \end{aligned} \tag{2.2}$$

The constants G_{Na} and G_K are the maximal conductance for sodium respectively potassium channels. An ion channel can be blocked, if its opening is covered by a so-called gating particle. Such gates are described in (2.2) by the time- and voltage dependent state-variables $m, n, h \in [0, 1]$. For very negative (positive) voltages, m and n approach 0 (1), thus closing (maximally opening) the corresponding channel. h approaches 0 (1) for very positive (negative) voltages. A channel is blocked, if all of its four gating particles are in the closed position. Figure 2.4 shows how m, h and n change with V_m during the generation of an action potential.

The generation of an action potential with its typical shape as shown in Figure 2.4 can now be understood as follows: before a stimulation, when $V_m \approx E_{rest}$, the membrane potential is basically determined by g_l and to a minor part by the slightly activated potassium channels. The sodium channels are almost fully closed. If the membrane becomes depolarized such that it approaches the threshold, m approaches 1 thus increasing g_{Na} causing an influx of sodium ions which further depolarizes V_m. This drives h towards 0 and n towards 1 thus closing the sodium channel and opening the potassium channel, allowing an outward current of K^+ ions. These two events hyperpolarize V_m, finally g_l brings V_m back to E_{rest}.

2.2.2 Dendritic Processing

Dendrites may be considered as the elementary computing devices in biological neural networks, where the input of an dendritic tree consists of temporal patterns of presynaptic spikes whereas the output is a potential change at the soma or at neighboring

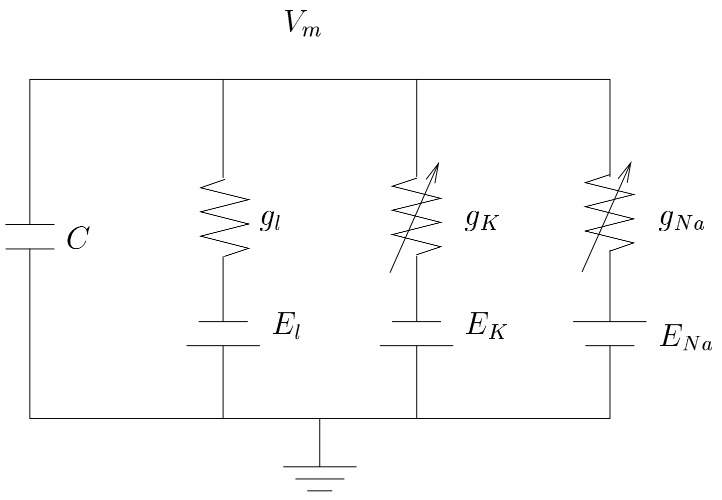

Figure 2.3: Equivalent circuit of an axonal membrane.

dendrites. Dendritic trees may vary considerably with respect to their geometry, their electrical properties and the synaptic input they receive.

Basic insights for a quantitative understanding of the integrative functioning of dendrites have been provided by Rall (a good collection of his papers can be found in [Segev et al., 1995]). In his *passive cable theory* he analyzes the spread of voltage in dendritic trees of arbitrary geometry in the absence of active, voltage-dependent ion channels.

The membrane of a dendrite is a relatively good insulator in comparison to the intra- or extracellular fluid. In the cable theory one considers small segments which can be approximated as cylinders and analyzes for such a cylinder the current flowing parallel to its axis x respectively the voltage $V = V(x,t)$ across the membrane at any point x and time t.

We consider now the current on a small membrane segment of such a cylinder. It consists of two components: the longitudinal current I_i can be described by Ohm's law:

$$\frac{1}{r_i}\frac{\partial V}{\partial x} = -I_i \tag{2.3}$$

where r_i is the resistance per unit length along the x-axis. I_i is taken positive when in the direction of increasing x values.

I_i changes along the x-axis due to the membrane current which charges (or discharges) the membrane capacitance or may flow through passive membrane channels. Then in the absence of additional currents the change per unit length of I_i is given by

$$\frac{\partial I_i}{\partial x} = -\left(\frac{V}{r_m} + c_m\frac{\partial V}{\partial t}\right) \tag{2.4}$$

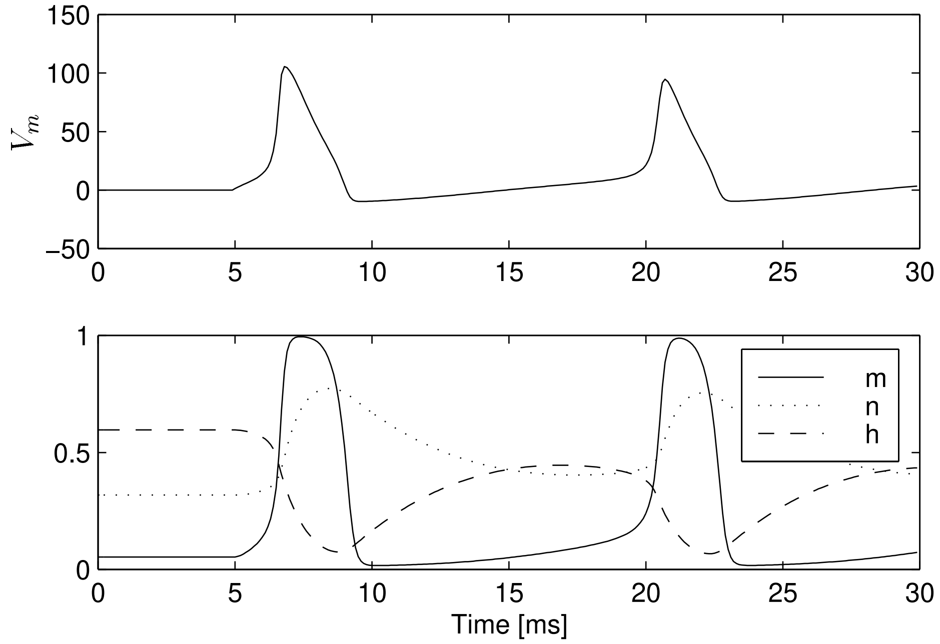

Figure 2.4: V_m during the generation of an action potential. Below the fractional activations of m, h and n are shown.

where r_m is the resistance across the membrane per unit length and c_m the capacitance across the membrane per unit length.

Putting (2.3) and (2.4) together we get the following *cable equation*:

$$\frac{1}{r_i}\frac{\partial^2 V}{\partial x^2} = c_m \frac{\partial V}{\partial t} + \frac{V}{r_m} \tag{2.5}$$

A complete and detailed derivation of the cable equation may be found e.g. in [Koch and Segev, 1992]. The solution of this equation depends strongly on the boundary conditions at the ends of the cylindrical segments. As shown by Rall it is possible to describe an arbitrary dendritic tree as a collection of such segments. Some of the major insights from this approach are:

- Voltage attenuates much more in the dendrites-to-soma direction than in the opposite direction.
- Synaptic potentials become wider as they come closer to the soma. Moreover, they are delayed depending on the position of the synapse on the dendritic tree. The delays make the dendrite sensitive to the temporal order of synaptic inputs. If the neuron receives excitatory input first from distal synapses and after that from

close excitatory synapses, the somatic depolarization is much stronger than if the input would have occurred in the reversed temporal order [Segev et al., 1995].

- Inhibitory synapses may hinder excitatory input when placed between the corresponding excitatory synapses and the soma. Hence their activation is specific to certain branches of the dendritic tree.
- Successive synaptic inputs do not sum linearly. The resulting postsynaptic polarization may be considerably smaller than the sum of the PSPs (see Chapter 6 in [Bower and Beeman, 1998]).
- If two EPSPs, caused by two synapses close to each other on the dendritic tree, overlap in time, then the resulting potential change at the soma will be less then the sum of the two EPSPs. This sublinearity is considerably reduced if the two synapses are placed on remote places of the dendritic tree (see [Segev et al., 1995]).

Recent experiments have shown that this last prediction of the cable theory is not always fulfilled: in [Cash and Yuste, 1998] it has been observed that pyramidal neurons in the hippocampus sum various EPSPs almost linearly, even if the corresponding synapses are placed very close to each other on the dendritic tree. Hence the spatial interaction of EPSPs is very well described by the LIFN model.

More detailed descriptions of the Hodgkin-Huxley model and dendritic processing can be found in [Koch and Bernander, 1995; Johnston and Wu, 1995; Bower and Beeman, 1998; Tuckwell, 1988].

2.3 Synaptic Functioning

The underlying mechanism for most chemical synapses can be summarized as follows: a chemical synapse – as shown in Figure 2.5 – has a gap of approximately 20 nm separating the axonal (presynaptic) endbulp from the postsynaptic membrane. The presynaptic endbulp contains Ca^{2+} ion channels, which are voltage dependent and which are closed in the absence of any inputs from the presynaptic neuron. If an action potential reaches the endbulp it causes the opening of these channels. The resulting Ca^{2+} influx initiates a movement of small vesicles, containing a neurotransmitter, to the membrane at the cleft, where the neurotransmitter is released and diffuses to the postsynaptic membrane. There it binds to receptors causing an opening of ion channels for a short time. The resulting ion influx leads to the postsynaptic potential.

As mentioned above, a synapse, which increases the probability of spike generation, is excitatory. If it reduces this probability, it is inhibitory. However, it is possible that an IPSP produces a depolarization. The two types of synapses are usually distinguished on the basis of their reversal potential E_{syn} in relation to the threshold Θ of the postsynaptic neuron: if $E_{syn} > \Theta$, then the synapse is excitatory, otherwise it is inhibitory. Hence, depending on the current membrane potential, an IPSP may depolarize the postsynaptic cell (see [Johnston and Wu, 1995] for details).

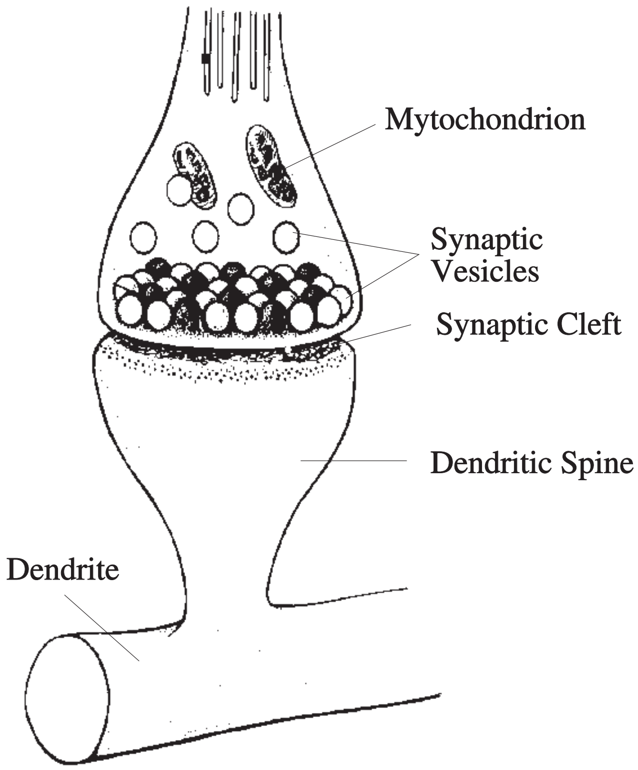

Figure 2.5: Typical synapse (after [Churchland and Sejnowski, 1992], p. 46).

There exist various types of neurotransmitters, each of them binds to a specific receptor type. One receptor which is very important for learning (e.g. for long term potentiation, see Section 2.4) is the N-methyl-D-aspartate (*NMDA*) receptor, which binds to the neurotransmitter glutamate. The activation of NMDA receptors causes the opening of ion channel which are mainly permeable to Ca^{2+}. Usually such a channel is blocked by magnesium (Mg^{2+}) ions. However, if the postsynaptic membrane is sufficiently depolarized, Mg^{2+} quits the channel thus allowing the influx of Ca^{2+}. Hence for the activation of such channels a presynaptic activity in connection to a postsynaptic depolarization is necessary. Such channels realize Hebbian learning, which will be characterized in the next section.

2.4 Synaptic Plasticity

There exists strong evidence that synaptic plasticity is a key factor for many learning phenomena. In the last few years there has been an enormous progress in understanding plasticity phenomena. This section reviews some of the main phenomena which will be of importance in the following.

In 1949 Donald O. Hebb addressed for the first time the question how synapses can modify their weights in order to store information. Hebb postulated his ideas in a time where no technology was available to study synaptic modifications. The situation has changed dramatically since the 1970s and nowadays there are lots of plasticity phenomena known which may be called "Hebbian". Hebb wrote ([Hebb, 1949], p. 62):

> When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased.

Hence such a *Hebbian synapse* is strengthened when the presynaptic and postsynaptic elements tend to be coactive. Formally the Hebb rule for the modification of the weight w_{AB} for the synapse from neuron A to B can be formulated as

$$\Delta w_{AB} = \eta \cdot V_A \cdot V_B$$

where $\eta > 0$ is the learning rate, determining the strength of learning. V_A and V_B denote the activity of neuron A respectively B, where it is in Hebbs original formulation not clear what is precisely meant by the term activity.

For Hebbian learning it is important that all information for the modification is *locally* available at the synapse and that the mechanism is time-dependent and interactive, i.e. the change depends on both the activity on the presynaptic and the postsynaptic side of the synapse. There are different formulations of the Hebb rule depending on the type of interaction they require. E.g. there exist conjunctive-type rules, simply requiring some simultaneous pre- and postsynaptic activity, but also correlational-type rules, where the synaptic enhancement depends on some statistical measure like the covariance between pre- and postsynaptic activity. Furthermore, there are Hebbian rules which consider only the time difference between single pre- and postsynaptic spikes.

Experimental evidence for Hebbian learning was given by the discovery of *long-term potentiation* (LTP), see e.g. [Brown et al., 1988]: neurons which were briefly but strongly stimulated in area CA1 of the hippocampus showed a rapid and sustained increase in synaptic efficacy if there was a simultaneous presynaptic neurotransmitter release and a postsynaptic depolarization. There exists also a counterpart to LTP, namely LTD (long-term depression) , which has been observed besides in the hippocampus also in the neocortex.

Only recently it has turned out that the synaptic efficacy may be influenced on the timing of single spikes: it was shown that the action potential in neocortical pyramidal cells is propagated backwards from the soma to the dendritic tree, such that information about the pre- and postsynaptic firing times is available at the synapse [Stuart and Sakmann, 1994]. In [Markram and Sakmann, 1995; Markram et al., 1997] evidence was given that the synaptic efficacy in such neurons can be actually changed due to this time difference: slice preparations of such neurons were investigated. Markram et al. were able to show that when a postsynaptic action potential and an EPSP coincided within a time window of 100 ms, changes in the synaptic efficacy were induced. After such a coupled stimulation, the EPSP amplitude was increased, if during the stimulation the EPSP preceded the postsynaptic action potential (by at least 10 ms). If the action potential occurred at least 10 ms before the EPSP the synaptic efficacy was reduced. These results suggests that single synapses may be up- or down-regulated in relation to the summed activity of all presynaptic neurons, which is reflected in the timing of the postsynaptic spikes.

In [Markram and Tsodyks, 1996; Tsodyks and Markram, 1996] it was shown that there is a redistribution of synaptic efficacy in neocortical pyramidal neurons. They observed that a synapse which received immediately after a coactivation of the corresponding pre- and postsynaptic neuron a short burst of presynaptic spikes changed its efficacy such that the first spike of a presynaptic burst was transmitted much more effectively than the subsequent spikes. The first spike causes a strong, fast rising PSP which drives the postsynaptic neuron very effectively. Taking into account that biological neurons can fire with a high precision of less than 1 millisecond (see [Mainen and Sejnowski, 1995]), such a neuron could reliably encode in its firing times certain features of its spatio-temporal input pattern.

2.5 Neural Coding

The last section has already touched one fundamental question in neurophysiology, namely how neurons encode information in their spike trains. One can distinguish essentially three different approaches:

- rate coding: the essential information is encoded in the firing rates, which are averaged over time or over several repetitions of the experiment. Sometimes one also considers a spatial rate, where an average over a population of neurons is taken.

- temporal coding: the timing of single spikes is used to encode information.

- population coding: information can be distinguished by the activity of different pools (populations) of neurons, where a neuron may participate at several pools.

This is only a very rough categorization and the dividing line between the various coding schemes cannot always be drawn clearly. Moreover, there are still ongoing debates about the question which neural codes are used by biological neural systems, there is growing evidence that the brain may use all three above-mentioned types and combinations of them. In the following we will focus on temporal coding which seems to be especially in the contents of fast information processing of importance. The relevant information may be represented in this case for example by the firing times of neurons relative to the stimulus onset [Gawne et al., 1996] such that the closer a neuron fires to the onset, the stronger the stimulation it signals. Hence only the first spike of a neuron carries relevant information. One might also assume that a neuron is shut off by some additional inhibitory input after its firing. Similarly one can imagine that information is encoded relative to the firing times of other neurons [O'Keefe and Reece, 1993; Thorpe and Imbert, 1996].

Evidence for the importance of the timing of the first spike has been given in [Thorpe et al., 1996]: they were able to show that humans can process visual patterns in 150 msec. Within this time it is hard to imagine that the neurons may sample firing rates, since there are only about 10 synaptic stages involved. Neurons participating in such computations usually have a firing rate of less than 100 Hz, hence 10 ms are not sufficient to estimate the current firing rate of some neuron [Thorpe and Imbert, 1989]. To use a population code here, one would need a huge amount of neurons to encode reliably sufficiently many variables (see [Thorpe et al., 1996] for details). Hence it is very likely that those computations are based on the timing of single spikes.

Further evidence for temporal coding has been given e.g. in [Sejnowski, 1995; Abeles, 1991; Abeles et al., 1993; Mainen and Sejnowski, 1995; Rieke et al., 1996].

3 A Computational Model for Spiking Neuron Networks

There exist various formal models for spiking neurons. The leaky integrate-and-fire neuron captures only some elementary features of biological neurons, but it is still able to reproduce a vast variety of properties of biological neural networks. It may be formally described by an equivalent circuit respectively by some corresponding equations (for a good description see Chapter 1 by Gerstner in [Maass and Bishop, 1998]). For our investigations the more general model of spiking neurons, which has been introduced in [Maass, 1996] is more suitable. We use the following definition of an *spiking neuron network* (SNN), which we cite from [Maass, 1996][1]:

Definition 3.1 *An SNN $\mathcal{N}$ consists of*

- *a finite directed graph $\langle V, E \rangle$ (we refer to the elements of V as "neurons" and to the elements of E as "synapses")*
- *a subset $V_{in} \subseteq V$ of input neurons*
- *a subset $V_{out} \subseteq V$ of output neurons*
- *for each neuron $v \in V - V_{in}$ a threshold function $\Theta_v : \mathbf{R}^+ \to \mathbf{R} \cup \{\infty\}$ (where $\mathbf{R}^+ := \{x \in \mathbf{R} : x \geq 0\}$)*
- *for each synapse $\langle u, v \rangle \in E$ a response function $\varepsilon_{u,v} : \mathbf{R}^+ \to \mathbf{R}$ and a weight $w_{u,v} \in \mathbf{R}^+$.*

We assume that the firing of the input neurons $v \in V_{in}$ is determined from outside of $\mathcal{N}$, i.e. the sets $F_v \subseteq \mathbf{R}^+$ of firing times ("spike trains") for the neurons $v \in V_{in}$ are given as the input of $\mathcal{N}$.

For a neuron $v \in V - V_{in}$ one defines its set F_v of firing times recursively. The first element of F_v is $\inf\{t \in \mathbf{R}^+ : P_v(t) \geq \Theta_v(0)\}$, and for any $s \in F_v$ the next larger element of F_v is $\inf\{t \in \mathbf{R}^+ : t > s$ and $P_v(t) \geq \Theta_v(t - s)\}$, where the potential function $P_v : \mathbf{R}^+ \to \mathbf{R}$ is defined by

$$P_v(t) := 0 + \sum_{\substack{u \in V \\ \langle u,v \rangle \in E}} \sum_{\substack{s \in F_u \\ s < t}} w_{u,v} \cdot \varepsilon_{u,v}(t - s)$$

[1]The corresponding definition in [Maass, 1996] allows in addition a time dependency of the weights, which is not necessary for the following.

(the trivial summand 0 makes sure that $P_v(t)$ is well-defined even if $F_u = \phi$ for all u with $\langle u, v\rangle \in E$). The firing times ("spike trains") F_v of the output neurons $v \in V_{out}$ that result in this way are interpreted as the output of $\mathcal{N}$.

The complexity of a computation in an SNN is evaluated by counting each spike as a computation step.

This very general definition is related to biological neurons as described in Chapter 2 in the following way: for a synapse $\langle u, v\rangle \in E$, the response-function $\varepsilon_{u,v}$ describes the shape of the PSP caused by an incoming spike to this synapse from neuron u. In the following we will assume that $\varepsilon_{u,v}(t) = 0$ for $t \in [0, d_{u,v}]$, where $d_{u,v}$ models the delay between the generation of the action potential at the axon hillock of neuron u and the time when the resulting PSP starts influencing the soma of neuron v. More precisely, this delay is given by $d_{u,v} = d_{u,v}^{pre} + d_{u,v}^{post}$, where $d_{u,v}^{pre}$ describes the time it takes for the action potential to propagate along the axon, finally causing the opening of the ion channels at the receptors of the postsynaptic membrane. $d_{u,v}^{post}$ is the time between the resulting ion influx and the the time when the potential of the soma of the postsynaptic neuron starts changing due to the PSP. It is basically given by the dendritic propagation time.

$P_v(t)$ corresponds to V_m of real neurons and describes the influence of the incoming EPSPs and IPSPs at the axon hillock of neuron v at time t. In the absence of any input to v, we set $P_v = 0$ corresponding to E_{rest}. Every $\varepsilon_{u,v}(t)$ is multiplied with a corresponding weight factor w_{uv}. This weight models the current strength of the synapse $\langle u, v\rangle$. Together with the choice for the response function it also takes into account the place of the synapse on the dendritic tree. $P_v(t)$ is computed by summing up the current values of all "relevant" PSPs, i.e. $w_{uv} \cdot \varepsilon_{u,v}(t - s)$ for all neurons u with $\langle u, v\rangle \in E$ which fired at a time s before t. Hence this definition does *not* take into account the non-linearities as described in Section 2.2.

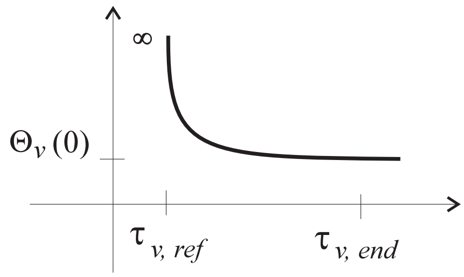

Figure 3.1: Typical shape of a threshold function. At $t = 0$ the neuron v generated an action potential

In Section 2.1 it was mentioned that after emitting a spike a neuron enters an absolute followed by a relative refractory period. In an SNN as defined above this can be modeled by considering for each neuron v two constants $\tau_{v,ref}$ and $\tau_{v,end}$ with $0 < \tau_{v,ref} < \tau_{v,end}$. One assumes that $\Theta_v(x) = \infty$ for $x < \tau_{v,ref}$ and that in the interval $[\tau_{v,ref}, \tau_{v,end}]$ the threshold function rapidly approaches a constant value $\Theta_v(0)$, such that $\Theta_v(x) = \Theta_v(0)$ for all $x > \tau_{v,end}$ (see Figure 3.1).

This model of an SNN may be considered as a noise free, deterministic version of the spike response model as introduced in [Gerstner, 1991; Gerstner et al., 1992]. In contrast to the spike response model, this definition of an SNN allows the investigation of different response- and threshold functions.

A definition of biologically more realistic models of neurons will be much more complex. One possibility to make more precise models of such neurons is provided by the compartmental modeling approach , which will be discussed in Chapter 9. For the following we will restrict ourselves to the above-given SNN-model.

4 On the Computational Power of SNNs

In this chapter we revise some of the main results about the computational power of SNNs presented in [Maass, 1996, 1997b] and then focus on the question whether the computational power depends on the shape of the postsynaptic potentials.

In [Maass, 1997b] SNNs are compared with artificial neural networks (ANNs). It is shown that SNNs are with regard to the number of neurons needed computationally more powerful than networks consisting of threshold gates respectively sigmoidal gates. There are functions, which may be also of relevance to biological systems, which can be computed by one spiking neuron, but when using an ANN with sigmoidal gates require a large number of hidden units.

Here we want to characterize the computational power of various kinds of SNNs by relating them to other computational models. For that purpose we employ the common notion of a *real-time simulation* from computational complexity theory (see e.g. [Leong and Seiferas, 1981; Maass, 1996]). One says that $\mathcal{M}'$ simulates $\mathcal{M}$ in real-time if $\mathcal{M}'$ can simulate each step in a computation of $\mathcal{M}$ with a *fixed* number of computation steps (i.e. the simulation of "later" computation steps of $\mathcal{M}$ does not require more steps of $\mathcal{M}'$ than the simulation of the first ones).

It is obvious that if $\mathcal{M}'$ simulates $\mathcal{M}$ in real-time, then it also simulates $\mathcal{M}$ in polynomial time (in fact: in linear time).

For biological neural systems the precise timing of computations is essential, and most computations are completed within a fixed number of "clock-cycles". Hence the notion of a real-time simulation is better suited for their investigation than the more common but too coarse notion of a polynomial simulation.

When we say in the following that a class $\mathcal{C}$ of machines can be simulated in real-time by SNNs with response- and threshold functions of a certain type (e.g. piecewise constant), we mean the following: we can construct for any machine $\mathcal{M}$ in $\mathcal{C}$ an SNN $\mathcal{M}'$ that simulates $\mathcal{M}$ in real-time, where *we* choose the architecture of $\mathcal{M}'$ as well as the values of delays $d_{u,v}$ and weights $w_{u,v}$ in $\mathcal{M}'$ and the "sign" of the response function (i.e. EPSP or IPSP). However, we allow that the exact shape of the response- and threshold functions of $\mathcal{M}'$ is *given* to us, i.e. they can be *arbitrary* functions of the specified type (e.g. piecewise constant) that satisfy certain conditions (see Section 4.1). Hence a simulation result of this type yields a real-time simulation of $\mathcal{M}$ with the simplest examples of such response- and threshold functions (see Figures 4.2 and 4.4), but also with any other response- and threshold functions that happen to satisfy the same conditions (see Figures 4.3 and 4.4). Thus a result of this type automatically implies that the exact shape of the response- and threshold functions of the considered type does not matter for this simulation result.

One fundamental result about the computational power of SNNs was given in [Maass, 1996], where it was shown that under certain, rather weak assumptions SNNs can simulate Turing machines in real-time:

Theorem 4.1 *One can construct an SNN $\mathcal{N}$ of finite size that can simulate with a suitable assignment of rational values from $[0,1]$ to its weights any Turing machine in real-time, if the response- and threshold functions employed for $\mathcal{N}$ satisfy certain assumptions (basically the EPSPs have to contain some small rising segment, see [Maass, 1996] for details).*

We will use the adjectives *analog* and *real-valued* interchangeably throughout this chapter. Parts of this chapter were presented in [Maass and Ruf, 1995, 1998].

4.1 SNNs with Piecewise Constant Response Functions

In the following two sections we focus on SNNs where the EPSPs and the IPSPs are described by piecewise constant functions. This is certainly the most elementary assumption from a mathematical point of view. We show that the computational power of such SNNs is indeed considerably weakened: they can no longer carry out the above-mentioned operations on analog variables in temporal coding. Thus there exists a significant difference in the computational power of spiking neurons with response functions of the types shown in Figures 4.1 and 4.2.

Besides our investigation of computations with analog inputs we also consider the case of computations of SNNs on bit strings, and show that their computational power drops from that of an arbitrary Turing machine to that of a deterministic finite automaton if their response functions are required to be piecewise constant instead of piecewise linear. In particular we will show that such SNNs can no longer carry out basic pattern matching operations in polynomial time.

SNNs use threshold functions in order to model the absolute and relative refractory periods of neurons. These functions describe for every neuron how its threshold varies after it has just fired. We show in this chapter that the shape of the threshold functions has much less influence on the computational power of SNNs.

Basic Definitions and Assumptions

As a formal model for an SNN we use Definition 3.1, where we make for the constructions in Sections 4.2 and 4.3 the following rather weak additional assumptions about the response- and threshold functions in an SNN.

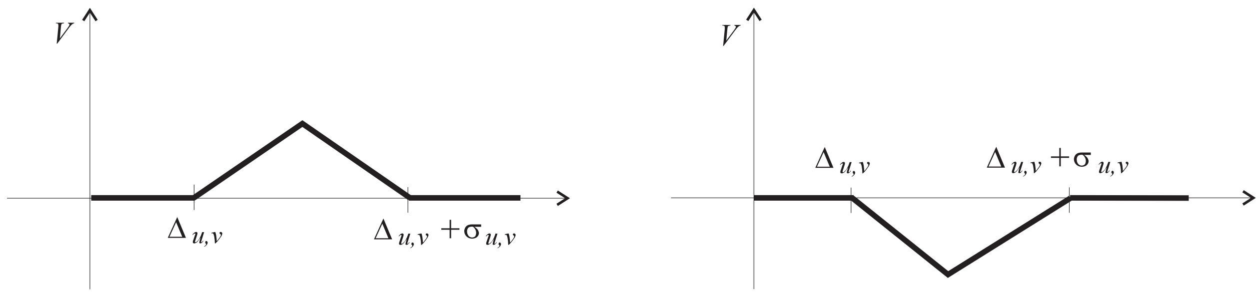

Figure 4.1: Simple piecewise linear response functions $\varepsilon_{u,v}$.

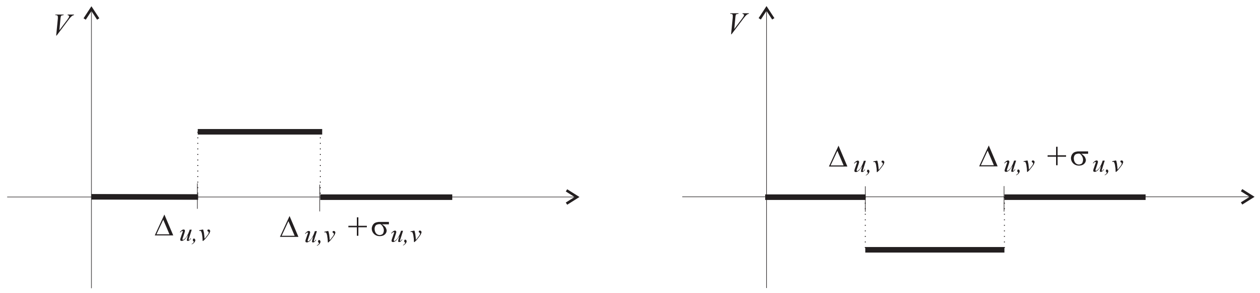

Figure 4.2: Simple piecewise constant response functions $\varepsilon_{u,v}$.

All response functions $\varepsilon_{u,v} : \mathbf{R}^+ \to \mathbf{R}$ and threshold functions $\Theta_u : \mathbf{R}^+ \to \mathbf{R}^+ \cup \{\infty\}$ are some arbitrary functions with the following properties: there exist constants $d_{min}, d_{max}, \sigma_{min}, \sigma_{max} \in \mathbf{R}^+$ with $0 < d_{min} < d_{max}$ and $0 < \sigma_{min} < \sigma_{max}$ such that the following conditions are fulfilled:

1. For every $\varepsilon_{u,v}$ there exists some delay $d_{u,v} \in [d_{min}, d_{max}]$ and, furthermore, some $\sigma_{u,v} \in [\sigma_{min}, \sigma_{max}]$ such that $\varepsilon_{u,v}(x) = 0$ for all $x \in [0, d_{u,v}] \cup [d_{u,v} + \sigma_{u,v}, \infty)$.

2. Every $\varepsilon_{u,v}$ either satisfies $\varepsilon_{u,v}(x) > 0$ for all $x \in (d_{u,v}, d_{u,v} + \sigma_{u,v})$ (in which case we refer to it as an EPSP) or $\varepsilon_{u,v}(x) < 0$ for all $x \in (d_{u,v}, d_{u,v} + \sigma_{u,v})$ (in which case we refer to it as an IPSP).

3. For every EPSP-response function $\varepsilon_{u,v}$ there exists some $\varepsilon_{u,v}^{max} > 0$ with $\varepsilon_{u,v}^{max} = \max\{\varepsilon_{u,v}(x) : x \in (d_{u,v}, d_{u,v} + \sigma_{u,v})\}$. Furthermore for all $x_1, x_2 \in (d_{u,v}, d_{u,v} + \sigma_{u,v})$ with $\varepsilon_{u,v}(x_1) = \varepsilon_{u,v}(x_2) = \varepsilon_{u,v}^{max}$ it follows that $\varepsilon_{u,v}(x) = \varepsilon_{u,v}^{max}$ for all $x \in [x_1, x_2]$.

4. For every $v \in V - V_{in}$ there exist constants $\tau_{v,ref}$, $\tau_{v,end} \in \mathbf{R}^+$, such that $\Theta_v(x) = \infty$ for all $x \in (0, \tau_{v,ref})$ (“absolute refractory period”), $0 < \Theta_v(0) \leq \Theta_v(x)$ for all $x \in [\tau_{v,ref} , \tau_{v,end})$ (”relative refractory period”), and $\Theta_v(x) = \Theta_v(0) > 0$ for all $x \geq \tau_{v,end}$.

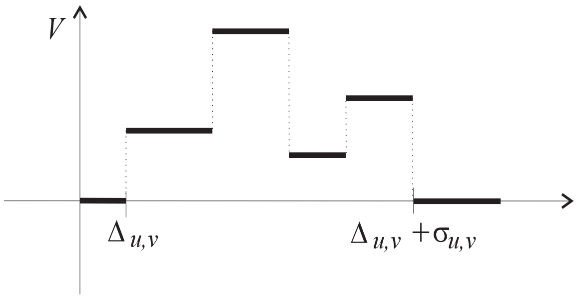

Figure 4.3: Example for a complicated piecewise constant response function $\varepsilon_{u,v}$ that satisfies our conditions 1 - 3.

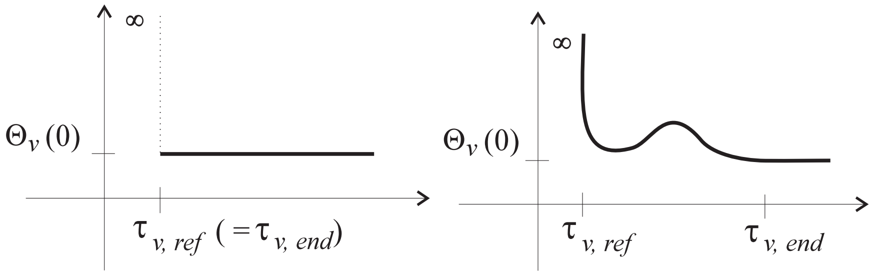

Figure 4.4: Left: Simple piecewise constant threshold function Θ_v. Right: Example for a more complicated piecewise monotone and continuous threshold function Θ_v that satisfies our condition 4.

We focus here on the computational power of SNNs with piecewise constant response functions $\varepsilon_{u,v} : \mathbf{R}^+ \to \mathbf{R}$ and piecewise monotone and continuous (respectively piecewise linear) threshold functions $\Theta_u : \mathbf{R}^+ \to \mathbf{R}^+ \cup \{\infty\}$. When considering piecewise constant response functions we assume that for every $\varepsilon_{u,v}$ there exist constants $m_{u,v} \in \mathbf{N}$ and $t_1^{u,v}, \ldots, t_{m_{u,v}-1}^{u,v} \in \mathbf{R}^+ \backslash \{0\}$ with $t_i^{u,v} < t_{i+1}^{u,v}, 1 \leq i \leq m_{u,v} - 2$ such that the domain $\mathbf{R}^+$ of that function can be partitioned into $m_{u,v}$ intervals $[0, t_1^{u,v}), [t_i^{u,v}, t_{i+1}^{u,v})$ with $1 \leq i \leq m_{u,v} - 2$ and $[t_{m_{u,v}-1}^{u,v}, \infty)$ such that $\varepsilon_{u,v}$ is constant on each of these intervals. We will choose $t_1^{u,v} = d_{u,v}$ and $t_{m_{u,v}-1}^{u,v} = d_{u,v} + \sigma_{u,v}$. A piecewise constant threshold function Θ_u is defined in the same fashion (see Figure 4.4).

In biological models one usually assumes in addition that the sign of the derivative of each response function $\varepsilon_{u,v}$ changes only once. It turns out that our negative results (i.e. lower bound results) even hold for the larger class of models where this assumption is *not* imposed, and hence we do not make this assumption in our formal model.

In the next section we will show that SNNs with piecewise constant response functions are real-time equivalent to a special type of random access machine (RAM), which we will call N⁻-RAM.

Definition 4.1 (N⁻-RAM)
An N⁻-RAM is a random access machine (RAM) with a constant number of registers that receives as its input, stores in its registers, operates on, and outputs real numbers from some bounded interval $[-B, B]$. The contents of a register R is denoted by $[R]$. The machine uses some designated register A as an accumulator. It can execute arbitrary programs of finite length, where each program statement has some unique label and consists of one of the following instructions:

ADD(β): *given some arbitrary constant $\beta \in [0, B]$, this command adds β to $[A]$ (provided that $[A] + \beta \in [-B, B]$) and stores the result in A.*

SUBTRACT(β): *given some arbitrary constant $\beta \in [0, B]$, this command subtracts β from $[A]$ (provided that $[A] - \beta \in [-B, B]$) and stores the result in A.*

IF COMPARE(R) THEN GOTO label: *this command compares the contents of the registers R and A. If $[A] \geq [R]$, then a jump to "label" is executed.*

GOTO label: *jumps to "label".*

LOAD(R): *loads the contents of R into A.*

STORE(R): *stores the contents of A into R.*

HALT: *unique instruction that ends the execution of the program.*

The input is given as the initial content of certain registers, and the output is given as the difference between the contents of two other distinguished registers when the machine halts. The complexity of a computation is evaluated according to the unit-cost criterion, where each execution of an instruction is counted as one computation step (regardless of the complexity of the operands).

Our output convention for N^--RAMs is motivated by the goal to prove that N^--RAMs and SNNs with piecewise constant response functions are real-time equivalent (Theorem 4.4). For SNNs we adopt the natural output convention that analog output values are represented as the difference between the firing times of two output neurons.

In [Maass, 1995] the stronger model of an N-RAM had been considered that can in addition execute the instructions ADD, SUBTRACT (on *two* arbitrary reals from some bounded interval), and MULTIPLY(β). This machine model was shown in [Maass, 1995] to be real-time equivalent to SNNs with piecewise linear activation functions and to recurrent analog neural nets with piecewise linear activation functions.

4.2 Characterization of the Power of Restricted SNNs for Analog Computations

Theorem 4.2 *Any SNN with piecewise constant response functions and piecewise monotone and continuous threshold functions can be simulated for real-valued input and output from a bounded range in real-time by an N^--RAM.*

PROOF We will show that for any given SNN $\mathcal{N}$ of the type considered here one can construct an N^--RAM $\mathcal{M}$ that can simulate $\mathcal{N}$ in real-time. The basic idea of the proof is that given the firing time t of some neuron in $\mathcal{N}$, the simulating N^--RAM $\mathcal{M}$ computes for each neuron v in $\mathcal{N}$ the *potential firing time* $t_v > t$, i.e. the first time where v would fire provided that no other neuron fires within the time-interval (t, t_v). From Definition 3.1 it follows that the neuron v for which t_v is minimal actually fires (there might in fact be several neurons that fire simultaneously at time t_v).

$\mathcal{M}$ reserves for each neuron in $\mathcal{N}$ a fixed number of registers. The firing times of the input neurons of $\mathcal{N}$ are assumed to be given to $\mathcal{M}$ as input in the form of the initial content of some registers. For each firing time t of an input neuron and for each later firing time t that it has already "constructed", $\mathcal{M}$ proceeds as follows: we assume that $\mathcal{M}$ stores for each neuron u all its firing times $\leq t$ which can still be relevant at times $\geq t$ for some other neuron v, i.e. which occured within the time-interval $(t-\sigma_{max}-d_{max}, t]$.[1] With $\tau_{min} := \min\{\tau_{u,ref} : u \in V\}$ it follows that at most $(\lfloor(\sigma_{max} + d_{max})/\tau_{min}\rfloor + 1) \cdot |V|$ registers are needed for that, since a neuron u can generate in a time interval of length $\sigma_{max} + d_{max}$ at most $\lfloor(\sigma_{max} + d_{max})/\tau_{u,ref}\rfloor + 1$ spikes.

We now show how $P_v(t)$ can be computed by $\mathcal{M}$: observe that $P_v(t)$ can assume only finitely many values, since v can receive only a bounded number of EPSPs respectively IPSPs which are still relevant at time t from some neuron u with $\langle u, v\rangle \in E$. With our assumption on the shape of the response functions it follows that each of them can contribute to $P_v(t)$ only one of finitely many values (of the form $w_{u,v} \cdot \varepsilon_{u,v}(t - t_u)$, with t_u being some firing time of neuron u).

In order to compute $P_v(t)$ at some time t, $\mathcal{M}$ has by definition to compute for every neuron u with $\langle u, v\rangle \in E$ and all firing times $t_u < t$ of u, which can be relevant to $P_v(t)$ the contribution $w_{u,v} \cdot \varepsilon_{u,v}(t - t_u)$ to $P_v(t)$. Therefore $\mathcal{M}$ has to find the largest $t_j^{u,v}$ (see Section 4.1) with $t_j^{u,v} \leq t - t_u$. Since $\varepsilon_{u,v}(t)$ is assumed to be piecewise constant, all possible values of $t_i^{u,v}$ and $w_{u,v} \cdot \varepsilon_{u,v}(t_i^{u,v})$ can be stored in a lookup-table (provided in the form of constants in certain registers of $\mathcal{M}$). If $j < m_{u,v} - 1$ then $\mathcal{M}$ also stores in some register the next time $t'_u = t_u + t_{j+1}^{u,v}$, when $P_v(t)$ changes again due to the firing of neuron u at t_u.

With the knowledge of $P_v(t)$ and of the first time t' when the potential changes

[1] It would be sufficient to consider the time-interval $(t - \sigma_{max,u} - d_{max,u}, t]$ for every neuron u with $d_{max,u} = \max\{d_{u,v} : v \in V\}$ and $\sigma_{max,u} = \max\{\sigma_{u,v} : v \in V\}$, but this gain in efficiency is irrelevant for our proof.

after t (given by the minimum of all the t'_u for all u with $\langle u,v\rangle \in E$), $\mathcal{M}$ can easily check whether there is some potential firing time during the time-interval $[t,t')$, i.e. a time where the potential function P_v meets the threshold function Θ_v. Since P_v can assume only finitely many values, it is possible to store in a lookup-table for every v and for every possible value P of P_v all $\tilde{t}$ for which $\Theta_v(\tilde{t}) = P$. If Θ_v has some constant segment of value P, $\mathcal{M}$ simply stores the corresponding interval-boundaries. Note that there are only finitely many values of $\tilde{t}$ to be stored due to our assumptions about the threshold function. Using this table $\mathcal{M}$ can find out whether there exists some $\tilde{t}$, such that $t_v+\tilde{t}$ is within $[t,t')$ and whether $\Theta_v(\tilde{t}) = P_v(t_v+\tilde{t})$ with t_v being the last firing time of neuron v (if v has not fired before t then we set $t_v = 0$ and replace $\Theta_v(\tilde{t})$ by $\Theta_v(0)$). If such $\tilde{t}$ can be found, then the smallest $\tilde{t}$ fulfilling this condition yields the next potential firing time $t_v + \tilde{t}$ of v. Otherwise $\mathcal{M}$ has to check iteratively whether the threshold is exceeded within the next constant segment of P_v.

Obviously it is sufficient to search for the potential firing times of v within the time-interval $[t, t+\sigma_{max}+d_{max})$ since no spike which occurred before t can have any influence on v at times $\geq t + \sigma_{max} + d_{max}$. As mentioned above the neuron with the smallest potential firing time actually fires. $\mathcal{M}$ stores this time in a register dedicated to this neuron and then continues its simulation in the same manner (taking this new firing time into account).

The registers containing (potential) firing times of $\mathcal{M}$ have to be kept bounded. Since only a bounded "time-window" of previous firings is relevant for determining future firings, M can subtract from the contents of all registers containing such firing times a suitable constant C and can erase those among these registers whose content is $< C$. ■

Theorem 4.3 *Any N$^-$-RAM can be simulated in real-time by an SNN with piecewise constant response- and threshold functions.*

PROOF We show that SNNs of the type considered here can store real numbers from some bounded interval with the help of oscillators, and that they can simulate within a bounded number of spikes every possible N$^-$-RAM instruction. The argument is based on the proof for Theorem 4.1 given in [Maass, 1996] for the real-time simulation of Turing machines by a less restricted class of SNNs.

In our SNN model, *oscillators* can be realized using two neurons u and v with $\langle u,v\rangle \in E$, $\langle v,u\rangle \in E$, $\varepsilon_{u,v}$ and $\varepsilon_{v,u}$ being EPSP-response functions. The weights are chosen such that $w_{u,v} \geq \Theta_v(0)/\varepsilon_{u,v}^{max}$ and $w_{v,u} \geq \Theta_u(0)/\varepsilon_{v,u}^{max}$. Once started, a spike "cycles" periodically through these two neurons. Such an oscillator has two inputs with which the oscillation can be started respectively halted, and one output, through which spikes with the oscillation period are sent out. These oscillators can be used in two ways for storing data: they can be used for storing single bits using their two states oscillating/dormant. Assuming the existence of a designated oscillator, which we call the *pacemaker* PM with oscillation period π_{PM}, any other oscillator O with the same oscillation period can also be used for representing real numbers modulo π_{PM} as the phase difference between O and PM. In order to represent negative numbers we assume that each oscillator representing some real number is associated with a second oscillator O_s representing the sign of that number. Numbers greater equal zero are represented by O as described above where O_s is dormant, for negative numbers we assume that O_s is oscillating with the same frequency and phase difference to PM as O. Note that for arbitrary $a \in \mathbf{R}^+$ we get $(-a \equiv \pi_{PM} - a) \bmod \pi_{PM}$. In order to represent for the given constant B of a given N$^-$-RAM all possible register contents $\in [-B, B]$, we assume that $\pi_{PM} > B$.[2] The oscillator corresponding to the accumulator A as described in Definition 4.1 will be denoted with O_A.

The program control can be realized in the same way as in [Maass, 1996], where it has been shown how SNNs can simulate arbitrary threshold circuits with boolean input, and thus simulate the control of some Turing machine in a very efficient way. This construction can also be applied for the type of SNNs considered here. A given N$^-$-RAM-program $\mathcal{P}$ can be described by a boolean function if we assume that each N$^-$-RAM-program statement is associated with a certain unique state. Each state can be described in a binary way using oscillators where a "1" ("0") is represented by an oscillating (dormant) state. We will refer to these oscillators as "state-oscillators". For every N$^-$-RAM-operation occurring in $\mathcal{P}$ we will construct one module which is started by the firing of some designated neuron, which acts as input neuron for that module. Thus there are as many ADD-modules as ADD-statements in $\mathcal{P}$ using different parameters (the same holds for the other parametrized statements). By using a layer of inter-neurons between the state-oscillators and those modules, it can be easily achieved that a certain state of the state-oscillators activates one unique module, executing the corresponding N$^-$-RAM-operation.

We now show how the N$^-$-RAM-instructions can be simulated by such SNN. We begin with the instruction COMPARE(R). A dedicated neuron u will fire within a specific time-interval if and only if $[A] < [R]$, where $[A]$ denotes the current content of the accumulator and $[R]$ the current content of register R. In order to realize the GOTO-part of the instruction, some straightforward circuitry makes sure that the binary states of the state-oscillators are reset to different values (representing the two possible next states of the N$^-$-RAM), depending on whether u fires within the specified time interval or not. In case that $A \geq 0$ and $R > 0$ we want to achieve that u fires if and only if the EPSP from u_2 (whose arrival time encodes $[A]$) reaches u before the IPSP from u_1 (whose arrival time encodes $[R]$). For that purpose one has to make sure that $w_{u_2,u}$ is sufficiently large so that in the absence of inhibitory input from other neurons the neuron u fires immediately when the EPSP from u_2 arrives at u. Furthermore, $w_{u_1,u}$ is chosen sufficiently large so that u will not fire if the IPSP from u_1 arrives before the

[2]If B is greater than the sum of the lengths of the time-intervals it takes $\varepsilon_{u,v}$ and $\varepsilon_{v,u}$ to reach $\varepsilon_{u,v}^{max}$ respectively $\varepsilon_{v,u}^{max}$, then one has to use a cycle of more than two neurons.

EPSP from u_2. In case that the length of this IPSP is too short to achieve this, one has to supplement it by IPSP's forming a chain of auxiliary inhibitory neurons that fire shortly after u_1. The binary states of the sign oscillators O_s for the accumulator A and the register R indicate whether the assumptions $[A] \geq 0$ and $[R] \geq 0$ for the case described above are actually met. If $[A] < 0$ and $[R] < 0$ one proceeds in an analogous manner to make sure that u fires within a specified time interval if $|[A]| > |[R]|$. If $[A] < 0$ and $[R] \geq 0$ the condition $[A] < [R]$ is always satisfied, and if $[A] > 0$ and $[R] < 0$ the condition $[A] < [R]$ is always false. Auxiliary circuitry makes sure that u fires in the former case, and does not fire in the latter case.

The simulation of the operation ADD(β) can basically be achieved by "sending" the spike of O_A through a delay module with delay β, i.e. a chain of neurons having only EPSP-links such that the delay of a spike passing through this chain adds up to β. In order to compute the sign of the addition, the result of that operation is temporarily stored in some oscillator O. Depending on the input $[A]$, $O_{A,s}$ has to be started/halted, which can be easily realized using a COMPARE-module. Finally the "content" of O is copied to O_A. SUBTRACT(β) can be realized in the same fashion.

The simulation of the LOAD(R)-operation signals to the oscillator representing $[R]$ to send a spike through its output to O_A. The delays have to be chosen such that O_A will actually represent $[R]$. We assumed that each LOAD-operation occurring in $\mathcal{P}$ corresponds to a unique state of the state oscillators. The proper register can be addressed in the same fashion as the different states of the state-oscillators address different modules. If $[R] < 0$ the corresponding sign-oscillators have to be copied as well.

The STORE(R)-operation can be realized in an analogous way by first halting the oscillator containing $[R]$ and then copying the "contents" of O_A to this oscillator. ∎

Theorems 4.2 and 4.3 immediately imply:

Theorem 4.4 *SNNs with piecewise constant response- and threshold functions are for computations with bounded real-valued input and output real-time equivalent to N^--RAMs.*

A closer look reveals that Theorems 4.2 and 4.3 together also imply the following result:

Corollary 4.1 *SNNs with piecewise constant response- and threshold functions are real-time equivalent to the class of SNNs with piecewise constant response functions and piecewise monotone and continuous threshold functions.*

It has been shown in [Maass, 1996] that an SNN that has small linearly increasing and decreasing segments in its response functions can add and subtract arbitrary bounded real numbers and also multiply arbitrary real bounded numbers with a given real constant. However, with piecewise constant response functions this is not possible, as the following theorem shows.

Theorem 4.5 *No SNN with piecewise constant response functions and piecewise monotone and continuous threshold functions can carry out with a bounded number of spikes any of the operations ADD, SUBTRACT, MULTIPLY(β) (for any $\beta > 0$ with $\beta \neq 1$) on arbitrary small differences in firing times between neurons. This holds even if the simulating SNN may employ arbitrary real-valued parameters.*

PROOF We prove by contradiction a slightly stronger result: no such SNN can *decide* with a bounded number of spikes for given (arbitrarily small) differences $a, b, c \geq 0$ in firing times between certain neurons whether $a + b = c$, $a - b = c$, or $a \cdot \beta = c$ (for any fixed $\beta > 0$ with $\beta \neq 1$). Assume that there exists an SNN of the considered type that solves any of these decision-problems with a bounded number of spikes. By Theorem 4.2 this implies that there exists an N$^-$-RAM $\mathcal{M}$ that can solve this decision problem for arbitrarily small inputs $a, b, c \geq 0$ with a bounded number of (say: at most ℓ) computation steps. Consider first the case where $\mathcal{M}$ decides whether $a + b = c$.

All possible computations of length $\leq \ell$ of this N$^-$-RAM $\mathcal{M}$ can be simulated by a decision tree T of depth ℓ with some rather special form of linear decision at its branching nodes. All register contents of $\mathcal{M}$ can be represented as a sum of at most one of the inputs a, b, c and a finite number of constants β. The linear decisions at the branching nodes of T represent applications of the instruction COMPARE to two sums of this type. Thus assuming that T contains m branching nodes, for every $1 \leq i \leq m$ the ith linear decision is of the form "$r_i + \gamma_i \geq s_i + d_i$", where $r_i, s_i \in \{a, b, c, 0\}$ and $\gamma_i, d_i \in \mathbf{R}^+$ are certain constants for this branching node.

Now we consider arbitrary inputs a, b, c with $0 < a < b < c < \epsilon/2$ with $\epsilon = \min\{|\gamma_i - d_i| : \gamma_i \neq d_i \text{ and } 1 \leq i \leq m\}$. It follows that each comparison "$r_i + \gamma_i \geq s_i + d_i$" in T holds for *all* such a, b, c if $\gamma_i > d_i$, and holds for *no* such a, b, c if $\gamma_i < d_i$ (if $\gamma_i = d_i$, its validity is predetermined by the pre-arranged order $a < b < c$). This results in a contradiction since the computation of T will arrive for all these inputs $\langle a, b, c \rangle$ at the same leaf of T (and hence give the same output), in spite of the fact that $a + b = c$ holds for some of these inputs, and does not hold for others.

Since $a - b = c$ holds if and only if $a = b + c$, the preceding argument automatically also covers the case of decisions "$a - b = c$". The argument for "$a \cdot \beta = c$" is analogous. ■

Corollary 4.2 *No SNN with piecewise constant response functions and piecewise monotone and continuous threshold functions is able to double through computations that involve at most a bounded number of spikes a difference in firing times between neurons, or a phase-difference between two oscillators (not even for arbitrarily small phase-differences).*

PROOF This follows directly from Theorem 4.5 (consider the operation MULTIPLY(2)). ■

4.3 Characterization of the Power of Restricted SNNs for Digital Computations

In this section we consider the case where the SNN receives an input $\underline{w} \in \{0,1\}^*$ in an online fashion, i.e. bit by bit, where $\{0,1\}^*$ is the set of all binary strings of finite length. We allow that the SNN signals through the firing of a designated neuron v_{prompt} that it wants to receive the next input bit. If the next input bit is "1", a designated neuron $v_{in} \in V_{in}$ will fire with a certain given delay $d \in \mathbf{Q}$ after the firing of v_{prompt}. If the next input bit is "0", v_{in} will not fire before the next firing of v_{prompt}.

The following theorem provides a stark contrast to Theorem 4.1:

Theorem 4.6 *SNNs with piecewise constant response functions and piecewise linear threshold functions with rational parameters are for online boolean input real-time equivalent to* *<u>d</u>eterministic <u>f</u>inite <u>a</u>utomatas (DFAs) .*

PROOF Assume that some SNN $\mathcal{N}$ as in the claim is given. Since $\mathcal{N}$ uses only rational parameters, the times t where the potential of some neuron changes or some neuron fires can be shown to be a multiple of some constant $d \in \mathbf{Q}$. We will represent the current state of all neurons of $\mathcal{N}$ at such time t as some state of the DFA $\mathcal{A}$ and compute by a transition-function the next state at time $t + d$.

The construction of the simulating DFA is carried out as follows: we will define a finite set of SNN-states in terms of the states of all neurons of $\mathcal{N}$ in such a way that every SNN-state has a unique successor-state. For that purpose it suffices if the state of some neuron v at time t contains the following information:

- the <u>spiking history</u> of v, given as the time difference between t and all firing times of v which can be still relevant to other neurons at times $\geq t$, i.e. all firing times of v which occur-ed within $(t - \sigma_{max} - d_{max},\ t]$.
- for every neuron u with $\langle u, v\rangle \in E$ and every firing time $t_u < t$ of u given by $t - t_u$ with $t - t_u \in (0,\ \sigma_{max} + d_{max})$, the <u>contribution of this spike to $P_v(t)$</u>, i.e. $w_{u,v} \cdot \varepsilon_{u,v}(t - t_u)$ and furthermore the <u>time-difference $t' - t$</u> between the smallest $t' > t$ (if any) and t with $\varepsilon_{u,v}(t' - t_u) \neq \varepsilon_{u,v}(t - t_u)$. The state of v also depends on the number i of the <u>current segment of $\varepsilon_{u,v}$</u>, with $1 \leq i \leq m_{u,v} - 1$ such that $t' - t_u = t^{u,v}_{i+1}$. If no t' exists then $i = m_{u,v}$.
- the <u>current threshold</u> $\Theta_v(t - t_v)$, where t_v is the last firing time of v. $t - t_v$ is given by the smallest element from the spiking history of v. If v has not fired before t or if $t - t_v \geq \tau_{v,end}$, then the current threshold is $\Theta_v(0)$.

As shown in the proof of Theorem 4.2, $P_v(t)$ can assume only finitely many values, which are in this case rational. For each possible value P of the potential function of some neuron v it follows that each t for which $P = \Theta_v(t)$ (if any) such that t is within a non-constant segment of Θ_v is rational, since the threshold functions were assumed to be piecewise linear. We denote all these possible times with $\theta_{v,1}, \ldots, \theta_{v,n_v}$ for some suitable constant $n_v \in \mathbf{N}$.

If a neuron $v \neq v_{in}$ fires at time t, then there has to be at least one neuron u, which fired at some time $t_u < t$ and which caused the "last jump" in the potential function of v before or at t. We can express t in terms of previous firing times: either $t = t_u + t_i^{u,v}$ for some i (i.e. the threshold was exceeded at a "jump" of the potential function) or $t = t_v + \theta_{v,j}$ for some j with t_v being the last firing time of v (i.e. the threshold was exceeded during a non-constant segment of the potential function). Obviously the latter case cannot occur if v did not fire within $[t - \tau_{v,end},\ t)$. Finally we observe that if the input neuron v_{in} fires at time t, then $t = t' + d$ for a firing time t' of the neuron v_{prompt}.

By induction it follows that for every neuron v any time t where v fires or its potential changes is rational and of the form

$$t = \sum_i n_i t_i + \sum_j \tilde{n}_j \theta_j + \hat{n} \cdot d \tag{4.1}$$

with $n_i, \tilde{n_j}, \hat{n} \in \mathbf{N}$, t_i of the form $\tilde{i}^{\tilde{u},\tilde{v}}$ and Θ_j of the form $\Theta_{\tilde{v},\tilde{j}}$ with $\tilde{i}, \tilde{j} \in \mathbf{N}$ and $\tilde{u}, \tilde{v} \in V$. The last term of equation 4.1 takes into account the delay d of the input neuron v_{in} after the firing of v_{prompt} (as described at the beginning of this section). Now we can easily choose a constant $d \in \mathbf{Q}$ such that for any such t there exists some $n \in \mathbf{N}$ with $t = n \cdot d$.

The preceding analysis implies that it is sufficient to consider $\mathcal{N}$ only at times $t = n \cdot d$ and that every neuron and thus also $\mathcal{N}$ can assume at those times only finitely many states. We model every state s of $\mathcal{N}$ as described above by a state s' of the DFA $\mathcal{A}$. Those states of $\mathcal{N}$ where v_{prompt} fires, will be mimicked by states of the DFA $\mathcal{A}$ where it reads its next input bit. A "1"-input causes the DFA to assume a state reflecting an SNN state where v_{in} fires at the corresponding time. Since there exists according to the preceding construction for each state of $\mathcal{N}$ and each time d a unique successor state of $\mathcal{N}$ at time $t + d$, we can define a corresponding transition function on states of $\mathcal{A}$ which allows $\mathcal{A}$ to simulate $\mathcal{N}$ for arbitrary online boolean input.

On the other hand a DFA can be simulated in real-time by an SNN of the type considered here in the same way as described in the proof of Theorem 4.3, since the simulation of boolean circuits on SNNs described there can be achieved using exclusively rational parameters. The states of the DFA are simulated by an array of oscillators in the SNN with binary states oscillating/dormant. ∎

An SNN of the type considered in Theorem 4.6, but with *real-valued* parameters, is computationally more powerful than a DFA, as the following corollary indicates:

Corollary 4.3 *Any Turing machine $\mathcal{M}$ can be simulated by an SNN with piecewise constant response functions and threshold functions (although not in real-time)*

PROOF In order to prove this result we have to design a mechanism which allows a *fixed* size SNN to store and manipulate bit sequences of *arbitrary* length.

It is well known that any Turing machine $\mathcal{M}$ can be simulated (but not in real time) by a counter machine $\mathcal{M}'$, having no tapes but two counters (see e.g. [Hopcroft and Ullman, 1979]). At each step $\mathcal{M}'$ can either increase or decrease one counter by one, or check if one counter is zero. An SNN can realize a counter with an oscillator O using the same idea as described in the proof of Theorem 4.3 by representing the current value of a counter as the phase-difference between O and some pace-maker π_{PM}. We choose a suitable constant β such that $k \cdot \beta = l \cdot \pi_{PM}$ for any $k, l \in \mathbf{N}$ implies that $k = l = 0$. Now the SNN can realize a counter incrementation (respectively decrementation) by using the ADD(β) and SUBTRACT(β) modules, as described in the proof of Theorem 4.3. In order to check if the counter is zero one can use the same idea as for the COMPARE-module. ■

The preceding result has shown that SNNs with piecewise constant response- and threshold functions can simulate arbitrary Turing machines, as it has been shown in Theorem 4.1 for SNNs with response functions that contain linearly increasing and decreasing segments. However, our next result exhibits an important difference between both classes of SNNs with regard to *speed* of these simulations. Whereas with the latter class of SNNs one can simulate arbitrary Turing machines in *real-time* (hence in *linear time*), *no* polynomial time simulation is possible if the response functions are piecewise constant.

Theorem 4.7 *Assume that a language $L \subseteq \{0,1\}^*$ is accepted in polynomial time by some online SNN $\mathcal{N}$ with arbitrary piecewise constant response functions and arbitrary piecewise monotone and continuous threshold functions, whose definition may involve arbitrary <u>real-valued</u> parameters. Then for every $n \in \mathbf{N}$ the initial segment $L \cap \{0,1\}^{\leq n}$ of L can be accepted by some DFA with at most polynomially in n many states.*

PROOF Theorem 4.4 also holds for online SNNs: one simply has to consider online N^--RAMs, which have in addition to the N^--RAM introduced in Definition 4.1 a READ-command, causing the next input-bit to be stored into some designated register.

An on-line N^--RAM $\mathcal{M}$ which simulates the given SNN $\mathcal{N}$ in real-time accepts $L \cap \{0,1\}^{\leq n}$ in at most polynomially in n many steps. The program of $\mathcal{M}$ is by definition of finite length and thus uses a finite number of constants. The possibilities of $\mathcal{M}$ to change the contents of registers are very limited (it can basically only add or subtract constants). Each of the (say k) registers of $\mathcal{M}$ can assume within polynomially in n many steps at most $p(n)$ different values for some polynomial p, independent from the input. Hence the registers of $\mathcal{M}$ can assume at most $p(n)^k$ "states" within polynomially in n many steps. Therefore a DFA with polynomially in n many states can simulate $\mathcal{M}$ for inputs up to length n, and hence accept $L \cap \{0,1\}^{\leq n}$. ■

Corollary 4.4 *No SNN of the type considered in Theorem 4.7 can decide in polynomial time whether* $\underline{w} = \tilde{w}$ *for two sequentially presented bit strings* $\underline{w}, \tilde{w} \in \{0,1\}^n$ *(i.e.* $\underline{w}\,\tilde{w}$*, or* $\underline{w}\#\tilde{w}$ *with a separation marker #, is given as input in an online fashion).*

PROOF By Theorem 4.7 it is sufficient to consider some DFA which carries out such a decision for a fixed n. It can be easily shown that such a DFA has to employ at least 2^n states to record the first half $\underline{w}$ of the input. ■

The pattern matching task from Corollary 4.4 can obviously be carried out by a Turing machine in linear time. Hence no SNN of the type considered in Theorem 4.7 and Corollary 4.4 can simulate an arbitrary Turing machine in polynomial time (i.e. in such a way that the simulation of t Turing machine steps requires at most polynomially in t many spikes). This provides a strong contrast to the situation for SNNs with linearly increasing and decreasing segments in their response functions, that can simulate any Turing machine in real-time (hence in linear time) even if all their parameters are rationals.

5 Elementary Computations with SNNs using Temporal Coding

There is not yet much known about possible computational mechanisms for SNNs which make use of the timing of single spikes and which may be also relevant for biological systems. First approaches have been given by Maass and Hopfield, which give rise to the simulation of artificial neural networks by SNNs using temporal coding [Hopfield, 1995; Maass, 1997a]. Hopfield's construction basically allows the simulation of radial basis functions where the relevant information (i.e. the centers of the radial basis functions) is stored in the delays between some neurons. It will be discussed in detail in Chapter 8.

One of the most common and important elements for ANNs is the sigmoidal gate. In [Maass, 1997a] it is shown how such gates can be simulated with SNNs. This construction, to which we will refer as the *Maass-construction* in the following, gives rise to a variety of computations with SNNs, which will be described in the following. In this chapter the aspects of the Maass-construction, which are relevant in the following, are given. We will restrict ourself to the case where no noise is present. As shown in [Maass, 1997a], the construction is very noise robust with respect to adding noise to the potential and the threshold of the neurons.

A sigmoidal gate with weights $w_i, 1 \leq i \leq m-1$ and "bias" w_m receives $m-1$ inputs $s_1, \ldots, s_{m-1} \in [0, \gamma]$ and computes $\sigma\left(\sum_{i=1}^{m-1} w_i s_i + w_m\right)$. The function $\sigma : \mathbf{R} \rightarrow [0, \gamma]$ is the activation function. It is called a sigmoidal function, if it is monotone, bounded (here to $[0, \gamma]$) and not constant.

The following construction can be formulated more easily if the bias is not considered separately. We assume that such a sigmoidal gate receives an additional constant input s_m, such that the function $f_G : \mathbf{R}^m \rightarrow [0, \gamma]$ computed by a sigmoidal gate can be written as

$$f_G(s_1, \ldots, s_m) = \sigma\left(\sum_{i=1}^{m} w_i s_i\right)$$

For most practical application it is sufficient to consider the following piecewise linear activation function $\pi_\gamma : \mathbf{R} \rightarrow [0, \gamma]$ which is defined as:

$$\pi_\gamma(y) = \begin{cases} 0, & \text{if} \quad y < 0 \\ y, & \text{if} \quad 0 \le y \le \gamma \\ \gamma, & \text{if} \quad y > \gamma \end{cases}$$

We want to simulate a π_γ-gate G which employs this activation function for some $\gamma > 0$ to be determined later and which computes a function $f_G : [0,\gamma]^m \to [0,\gamma]$ defined by:

$$f_G(s_1,\ldots,s_m) = \begin{cases} 0\,, & \text{if} \quad \sum_{i=1}^m w_i s_i < 0 \\ \sum_{i=1}^m w_i s_i, & \text{if} \quad 0 \le \sum_{i=1}^m w_i s_i \le \gamma \\ \gamma\,, & \text{if} \quad \sum_{i=1}^m w_i s_i > \gamma \end{cases}$$

In the next section we will describe how the main ingredient, namely the computation of a weighted sum, is achieved with SNNs in temporal coding by the Maass-construction.

5.1 Computing Weighted Sums

The main idea is to consider a spiking neuron v which receives input from $m+1$ spiking neurons $u_0,\ldots,u_m$. All these input neurons fire exactly once within a sufficiently small time interval, where u_0 produces a reference spike and neuron u_i, $1 \le i \le m$ encodes through its firing the input value s_i. These spikes make v fire at a time which is basically determined by $\sum_i w_i s_i$. The main assumption here is as in Chapter 4 that the initial segments of the PSPs caused by the input neurons rise linearly (see Figure 5.1).

More precisely, we want v to compute a function $f_G : [0,\gamma]^m \to [0,\gamma]$ defined by:

$$f_G(s_1,\ldots,s_m) = \begin{cases} \sum_{i=1}^m w_i s_i, & \text{if } 0 \le \sum_{i=1}^m w_i s_i \le \gamma \\ \text{arbitrary}, & \text{otherwise} \end{cases}$$

for some $\gamma > 0$ to be determined later.

Given an input vector $\mathbf{s} = (s_1,\ldots,s_m)$. We assume that each input neuron u_i, $0 \le i \le m$ fires exactly once within the time-interval $[T_{in}-\gamma, T_{in}]$ to which we will also refer as the *coding interval*. Neuron u_0 fires at time $t_0 = T_{in}$, neuron u_i, $1 \le i \le m$ fires at time $t_i = T_{in} - s_i$, thus representing the input s_i in temporal coding relative to the reference neuron u_0. The neuron u_0 represents the value $s_0 = 0$.

As in Chapter 4 we assume that the firing of a neuron u_i causes after a delay of length d (which is equal for all input neurons) a PSP for neuron v which has an initial linear segment of length Δ. In case of an EPSP (i.e. $w_i \ge 0$) we have

$$\varepsilon_{u_i,v}(t) = \begin{cases} 0, & \text{for } t \in [0,d] \\ t-d, & \text{for } t \in [d,d+\Delta] \\ \text{arbitrary}, & \text{for } t > d+\Delta \end{cases} \tag{5.1}$$

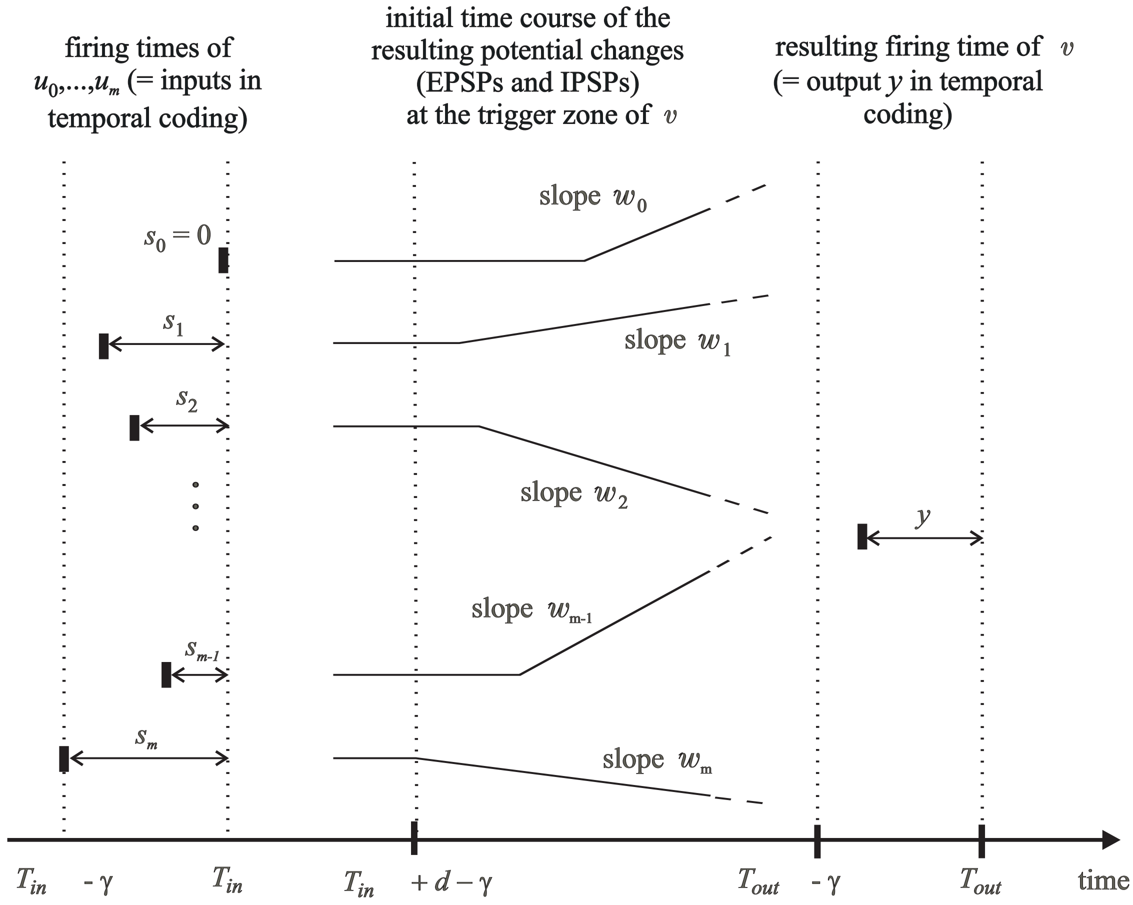

Figure 5.1: Computation of a weighted sum with a spiking neuron (from [Maass, 1997a]).

whereas in case of an IPSP (i.e. $w_i < 0$) the response function is of the type

$$\varepsilon_{u_i,v}(t) = \begin{cases} 0, & \text{for } t \in [0,d] \\ -(t-d), & \text{for } t \in [d, d+\Delta] \\ \text{arbitrary}, & \text{for } t > d+\Delta \end{cases}$$

We want to achieve that v fires at a time t_v where all incoming PSPs are in their linear segment, i.e. where

$$d \leq t_v - (T_{in} - s_i) \leq d + \Delta \qquad \text{for } i = 0, \ldots m \tag{5.2}$$

The potential of v is similar as in Definition 3.1 given by

$$P_v(t) := \sum_{i=1}^{m} \sum_{\substack{s \in F_{u_i} \\ s<t}} |w_i| \cdot \varepsilon_{u_i,v}(t-s) \tag{5.3}$$

where F_{u_i} denotes again the set of firing times of u_i prior to t. Hence if 5.2 is fulfilled the firing time t_v is determined by

$$\Theta = \sum_{i=0}^{m} w_i(t_v - (T_{in} - s_i) - d) \tag{5.4}$$

As mentioned above, we set $s_0 = 0$. We choose the corresponding weight w_0 such that $\sum_{i=0}^{m} w_i = \lambda$. The constant $\lambda > 0$ is chosen such that v actually fires (it is also of importance in the presence of noise, see [Maass, 1997a] for details). We can now rewrite (5.4) as

$$t_v = \frac{\Theta}{\lambda} + T_{in} + d - \frac{\sum_{i=1}^{m} w_i s_i}{\lambda} \tag{5.5}$$

Since $s_j \in [0, \gamma]$ for $j = 0, \ldots, m$, we can rewrite (5.2) as

$$0 \leq \frac{\Theta - \sum_{i=1}^{m} w_i s_i}{\lambda} \leq \Delta - \gamma \tag{5.6}$$

If this condition is fulfilled, the firing time depends - up to several constants - only on $\sum w_i s_i$. We introduce a new constant

$$T_{out} := \frac{\Theta}{\lambda} + T_{in} + d$$

such that (5.5) can now be rewritten as

$$t_v = T_{out} - \frac{1}{\lambda} \cdot \sum_{i=1}^{m} w_i s_i \tag{5.7}$$

Since we assumed that $\sum_{i=1}^{m} w_i s_i \in [0, \gamma]$, condition (5.6) can be formulated as

$$\gamma \leq \frac{\Theta}{\lambda} \leq \Delta - \gamma \tag{5.8}$$

Hence, γ and λ have to be chosen according to this condition in order to guarantee that t_v temporally encodes the output $f_G(s_1, \ldots, s_m) = \sum_{i=1}^{m} w_i s_i$ for $\sum_{i=1}^{m} w_i s_i \in [0, \gamma]$.

5.2 Simulating Sigmoidal Gates

For simulating a π_γ-gate we have to extend this construction such that v fires at time T_{out} if $\sum_{i=1}^{m} w_i s_i < 0$ and at time $T_{out} - \gamma$ if $\sum_{i=1}^{m} w_i s_i > \gamma$. This can be achieved (up to an error of a given ε with $\gamma > \varepsilon > 0$) by using additional inhibitory and excitatory neurons. The inhibitory neurons inhibit v from firing before $T_{out} - \gamma - \varepsilon$, the excitatory neurons guarantee that v fires no later than T_{out}. The firing of these auxiliary neurons does not depend on the input $\mathbf{s}$, only on T_{in}. For details we refer to [Maass, 1997a]. The resulting activation function is shown in Figure 5.2.

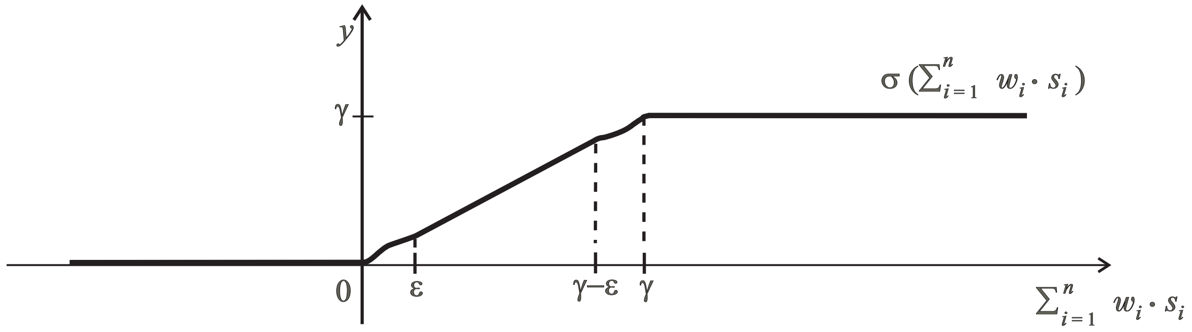

Figure 5.2: The activation function computed by a spiking neuron realizing the "Maass-construction" (from [Maass, 1997a]).

We have seen that the neuron v can approximate a π_γ gate. Since both the inputs and the output use the same coding scheme, it is possible to consider a layer of such neurons forming the input to another layer of such neurons. Hence it is possible to simulate feedforward networks consisting of sigmoidal gates.

Feedforward sigmoidal nets with one layer of hidden units are universal approximators , i.e. they can approximate any given bounded continuous function $F : [0, \gamma]^n \to [0, \gamma]^k$ with any degree of accuracy. In [Leshno et al., 1993] it is shown that the precise form of the activation function is irrelevant as long as it is not a polynomial. Hence it is possible to use the approximation of π_γ-gates described above for realizing such networks through SNNs using temporal coding. The details of this construction are given in [Maass, 1997a].

6 Supervised Learning with SNNs

The following chapters will discuss various learning mechanisms for spiking neurons using temporal coding. As mentioned in Section 2.4, weight changes for some learning process are often realized according to the Hebb rule, where a synapse is strengthened if both the presynaptic and postsynaptic neuron have at the same time a high firing rate. Since we focus here on single firing events, we consider a variation of the Hebb rule, where the weight change depends on the time difference of single presynaptic and postsynaptic firing times. This type of learning is motivated by recent neurobiological results as described in Section 2.4.

In this chapter we will be concerned with supervised learning, where some error signal is available. Many learning algorithms for ANNs are supervised as the very popular error-backpropagation algorithm, which will be discussed in short in Chapter 10. Here we focus on a very fundamental question arising in temporal coding namely how is the relation between weights and temporally encoded inputs, i.e. how can a synapse adopt its strength to a certain value. More precisely, the goal is that the synapse changes its weight such that a presynaptic spike makes the postsynaptic neuron fire after a predetermined time. The supervision is provided by a second presynaptic spike indicating when the postsynaptic neuron should have fired due to the first presynaptic spike. Then the weight is modified due to the error signal given by the time difference between the second presynaptic and the postsynaptic spike.

Thorpe and Gautrais have recently shown that the relation between weights and spike timing could be crucial for fast information processing of neurons in the visual system [Thorpe and Gautrais, 1997]. They demonstrated that feature extraction might be achieved by the visual system using the order of firing and a proper tuning of synaptic strengths. Hence the question arises, how a synapse can modify its weight simply on the basis of time differences between firing times and not on firing rates, or more generally formulated, how a neural system can store information about certain stimuli which are presented repeatedly in the same temporal coding.

This question is also relevant in the context of the Maass-construction presented in Chapter 5. If one wants to teach a sigmoidal gate to compute a certain function, one possibility is to present in a supervised way the desired weight-values to the neuron.

In the following two sections we will present two possible solutions to this question, which achieve that the weights of a neuron assume certain values and where the weight changes during the learning process are only based on differences between pre- and postsynaptic firing times. Section 6.1 focuses on the case when only one synapse learns at a time. Section 6.2 discusses how a similar result may be achieved when all synapses of a neuron learn in parallel.

Excerpts of Section 6.1 and 6.2 appeared in [Ruf and Schmitt, 1997a,b].

6.1 Monosynaptic Learning

In this section we introduce a Hebbian-type learning rule for a single synapse which is based solely on single pre- and postsynaptic spikes. The synapse receives two presynaptic spikes. The goal is that the synaptic weight assumes a value determined by the time difference of these two spikes.

As in Chapters 4 and 5 we assume that the initial segments of the PSPs rise linearly. To simplify our notation, we make in addition the following assumption: if a neuron u, having an excitatory connection to neuron v, fires then a spike arrives at $\langle u, v\rangle$ after a delay $d_{u,v}^{pre}$. The synapse $\langle u, v\rangle$ does not perceive the actual firing but the arrival of the spike instead. Since we will discuss local learning rules for such synapses, we will describe firing times of u in terms of arrival times at $\langle u, v\rangle$ throughout this chapter. Hence when speaking of a firing time t_u, we actually mean the arrival time of this spike at the synapse $\langle u, v\rangle$, thus neglecting the delay $d_{u,v}^{pre}$. The same simplification is made for the dendritic delays. We would like to emphasize that these assumptions are only made to keep arguments simple, the constructions of this chapter also hold without them.

Our construction works as follows: neuron u fires at a certain time t_u making v fire at t_v. u fires at a second time $t_0 > t_u$ which is essential for learning in the monosynaptic model in two ways. First, together with t_u it defines a time interval $[t_u, t_0]$ having length $t_0 - t_u$ which can be considered as the input value for v at $\langle u, v\rangle$. Second, together with t_v it will be used in the learning rule that depends only on these two firing times.

We suggest the following weight update as a monosynaptic, Hebbian-type learning rule:

$$\Delta w_{u,v} = \eta(t_v - t_0) \tag{6.1}$$

where $\eta > 0$ is the learning rate, a parameter that has to be chosen before learning starts but is fixed during all learning cycles. The time difference $t_v - t_0$ between the presynaptic spike of u at t_0 and the postsynaptic spike of v at t_v can be considered as an error that has to be reduced during learning. The learning rule is applied once during a certain period of time which we call a learning cycle. The spike trains that are emitted from u and v during such a cycle are shown in Figure 6.1.

The main result of this section is a proof that by repeating the learning cycles $w_{u,v}$ approaches a value $\widetilde{w}$ determined by the two input spikes. It will turn out that $\widetilde{w}$ is proportional to $1/(t_0 - t_u)$.

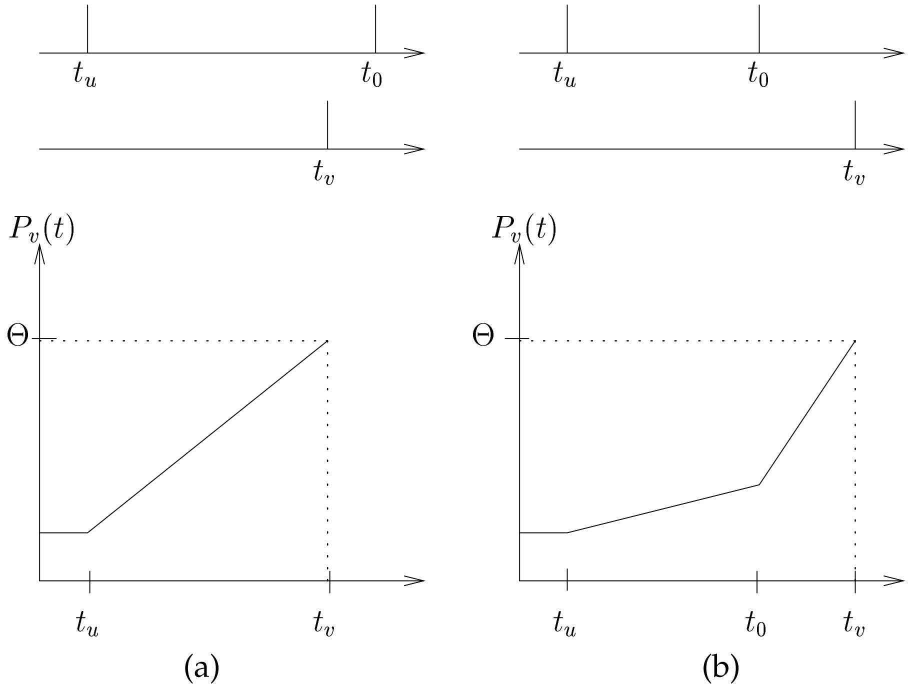

Figure 6.1: Spikes at synapse $\langle u, v\rangle$ and potential of neuron v during a learning cycle. In (a) the first spike of u is sufficient to make v fire before the second spike of u; in (b) the firing of v is determined by two spikes of u.

Theorem 6.1 *Given two neurons u, v with excitatory synapse $\langle u, v\rangle$ having a response-function $\varepsilon_{u,v}$ fulfilling the linearity-assumption (5.1). Let the interval $[w_{\min}, w_{\max}] \subseteq \mathbf{R}^+$ be the range of possible values for weight $w_{u,v}$ and the learning rate η be any arbitrary real number satisfying $0 < \eta \leq w^2_{\min}/(\Theta - P_v^{rest})$ where P_v^{rest} denotes the resting membrane potential of v, i.e. when it receives no input. If neuron u fires twice during each learning cycle with fixed time difference $t_0 - t_u$ and $w_{u,v}$ is updated according to rule (6.1) with learning rate η then, for any arbitrary initial weight from $[w_{\min}, w_{\max}]$, the sequence of weight values converges to $\widetilde{w}$ such that $\widetilde{w} = (\Theta - P_v^{rest})/(t_0 - t_u)$.*

PROOF First we show that the only fixed point of the learning rule is

$$\widetilde{w} = (\Theta - P_v^{\text{rest}})1/(t_0 - t_u). \tag{6.2}$$

The application of (6.1) does not change $w_{u,v}$ if and only if the second firing of u at t_0 and the firing of v at t_v take place simultaneously, i.e. $t_v = t_0$. With the linearity assumption for the EPSPs we get

$$P_v^{\text{rest}} + \widetilde{w}(t_v - t_u) = \Theta.$$

Substituting t_0 for t_v we obtain

$$t_0 - t_u = \frac{\Theta - P_v^{\mathrm{rest}}}{\widetilde{w}} \tag{6.3}$$

Thus, the fixed point $\widetilde{w}$ of the learning rule is proportional to $1/(t_0 - t_u)$. which is equivalent to (6.2).

It remains to show that any $w \in [w_{\min}, w_{\max}]$ converges to $\widetilde{w}$ by an iterated application of (6.1), i.e. that $\widetilde{w}$ is also an attracting fixed point for these weights. Let w be the current value of $w_{u,v}$ where $w \neq \widetilde{w}$. We distinguish between two cases, $w > \widetilde{w}$ and $w < \widetilde{w}$. This is necessary because if $w > \widetilde{w}$ then the firing of v at t_v takes place before the second firing of u at t_0. Hence, t_v depends only on $P_v^{\mathrm{rest}}, \Theta$, and t_u. In the case $w < \widetilde{w}$ neuron u fires twice, at t_u and t_0, before $P_v(t)$ reaches Θ. Hence t_v depends also on t_0. We show that in both cases the value of $w_{u,v}$ stays on the same side of $\widetilde{w}$ and approaches $\widetilde{w}$ successively.

Case 1: $w > \widetilde{w}$. In this case, which is shown in Figure 6.1(a), the firing of v at t_v satisfies

$$P_v^{\mathrm{rest}} + w(t_v - t_u) = \Theta$$

i.e. t_v is given by

$$t_v = \frac{\Theta - P_v^{\mathrm{rest}}}{w} + t_u.$$

Together with (6.3) we obtain from this an expression for the change of w resulting from a single application of rule (6.1) by

$$\Delta w = \eta(t_v - t_0) = \eta\left(\frac{\Theta - P_v^{\mathrm{rest}}}{w} - \frac{\Theta - P_v^{\mathrm{rest}}}{\widetilde{w}}\right).$$

Thus we can describe the progression of w by an iterative map $f : \mathbf{R} \to \mathbf{R}$ defined as

$$f(w) = w + \kappa\left(\frac{1}{w} - \frac{1}{\widetilde{w}}\right) \quad \text{with } \kappa = \eta(\Theta - P_v^{\mathrm{rest}}).$$

It is obvious that $\widetilde{w}$ is a fixed point of f. It remains to show that f maps $[\widetilde{w}, w_{\max}]$ into itself and is contractive on $[\widetilde{w}, w_{\max}]$, i.e.

$$|f(x) - f(y)| \leq q \cdot |x - y|, \qquad \text{for all } x, y \in [\widetilde{w}, w_{\max}]$$

for some $0 \leq q < 1$. Then it follows by a well known fixed point theorem (see, e.g., [Blum, 1972, Theorem 5.1.1]) that f has a unique fixed point in $[\widetilde{w}, w_{\max}]$, which is then $\widetilde{w}$, and that the sequence $w^{(n+1)} = f(w^{(n)})$ converges to that fixed point for any choice of $w^{(0)} \in [\widetilde{w}, w_{\max}]$.

First we show $f(w) \geq \widetilde{w}$ for all $w \in [\widetilde{w}, w_{\max}]$. This is equivalent to

$$w + \kappa/w \geq \widetilde{w} + \kappa/\widetilde{w} \quad \text{for all } w \in [\widetilde{w}, w_{\max}]. \tag{6.4}$$

The condition for the learning rate $\eta \leq w_{\min}^2/(\Theta - P_v^{\text{rest}})$ can be rewritten as $w_{\min}^2 \geq \kappa$ from which

$$1 - \frac{\kappa}{w^2} \geq 0 \quad \text{for all } w \geq w_{\min} \tag{6.5}$$

follows. The expression $1 - (\kappa/w^2)$ is the derivative of $g(w) = w + (\kappa/w)$ which is therefore increasing on $[w_{\min}, w_{\max}]$. Hence (6.4) is proven and $f(w) \geq \widetilde{w}$ holds for all $w \in [\widetilde{w}, w_{\max}]$.

From $w > \widetilde{w}$ we get

$$\kappa \left(\frac{1}{w} - \frac{1}{\widetilde{w}} \right) < 0.$$

Thus $f(w) < w$ holds which implies $f(w) < w_{\max}$ for all $w \in [\widetilde{w}, w_{\max}]$. It follows from this, together with $f(w) \geq \widetilde{w}$, that f maps $[\widetilde{w}, w_{\max}]$ into itself.

To see that f is contractive on $[\widetilde{w}, w_{\max}]$ we consider f', the derivative of f, and show that $|f'(w)| \leq L$ for all $[\widetilde{w}, w_{\max}]$ with Lipschitz constant $L < 1$. The derivative of f is

$$f'(w) = 1 - \frac{\kappa}{w^2}.$$

In (6.5) this expression was already seen to be non-negative. A Lipschitz constant $L < 1$ such that $f'(w) \leq L$ for all $w \in [\widetilde{w}, w_{\max}]$ is then provided by $L = 1 - (\kappa/w_{\max}^2)$.

Case 2: $w < \widetilde{w}$. Here the potential of neuron v is determined by two spikes of neuron u, at t_u and at t_0 (see Figure 6.1(b)). The firing time t_v of v then satisfies

$$P_v^{\text{rest}} + w(t_v - t_u) + w(t_v - t_0) = \Theta$$

from which we derive

$$t_v = \frac{1}{2} \left(\frac{\Theta - P_v^{\text{rest}}}{w} + t_0 + t_u \right).$$

Solving (6.3) for t_0 and substituting we can rewrite this as

$$t_v = \frac{1}{2} \left(\frac{\Theta - P_v^{\text{rest}}}{w} + \frac{\Theta - P_v^{\text{rest}}}{\widetilde{w}} \right) + t_u.$$

Combined with (6.3) this yields a synaptic change of

$$\Delta w = \eta(t_v - t_0) = \frac{\eta}{2} \left(\frac{\Theta - P_v^{\text{rest}}}{w} - \frac{\Theta - P_v^{\text{rest}}}{\widetilde{w}} \right).$$

The sequence of weights is then determined by iterating the function $h : \mathbf{R} \to \mathbf{R}$ defined as

$$h(w) = w + \lambda \left(\frac{1}{w} - \frac{1}{\widetilde{w}} \right) \quad \text{with } \lambda = \eta(\Theta - P_v^{\text{rest}})/2.$$

Obviously, $\widetilde{w}$ is a fixed point of h. It remains to show that h maps $[w_{\min}, \widetilde{w}]$ into itself and is contractive on $[w_{\min}, \widetilde{w}]$. From $\lambda = \kappa/2$ and (6.5) we get

$$w + \lambda/w \leq \widetilde{w} + \lambda/\widetilde{w} \quad \text{for all } w \in [w_{\min}, \widetilde{w}]$$

which is equivalent to $h(w) \leq \widetilde{w}$ for all $w \in [w_{\min}, \widetilde{w}]$. From $w < \widetilde{w}$ we get $\lambda(1/w - 1/\widetilde{w}) > 0$ and hence $h(w) > w_{\min}$ for all $w \in [w_{\min}, \widetilde{w}]$. Thus h maps $[w_{\min}, \widetilde{w}]$ into itself. Finally, along the same lines of reasoning as in case 1, it is now easy to conclude that h is contractive on $[w_{\min}, \widetilde{w}]$ with Lipschitz constant $L = 1 - (\lambda/w_{\max}^2) < 1$.

This completes the proof of the theorem. ∎

With regard to the convergence rate we can make the following observation: It is well known that for contractive iterative mappings the rate of convergence is governed by

$$|w^{(n)} - \widetilde{w}| \leq L^n |w^{(0)} - \widetilde{w}|$$

where $L < 1$ is a Lipschitz constant [Blum, 1972]. Thus the sequence of weights generated according to the learning rule (6.1) satisfies

$$|w^{(n)} - \widetilde{w}| \leq \left(1 - \frac{\mu}{w_{\max}^2}\right)^n |w^{(0)} - \widetilde{w}|$$

where

$$\mu = \begin{cases} \eta(\Theta - P_v^{\text{rest}}), & \text{if } w^{(0)} > \widetilde{w} \\ \eta(\Theta - P_v^{\text{rest}})/2, & \text{if } w^{(0)} < \widetilde{w} \end{cases}$$

In the theorem we put no restrictions on values $w_{\min}$ and $w_{\max}$, except that they have to be positive, in order to keep the statement and the proof as general as possible. However, within biologically more realistic formulations, certain restrictions are necessary:

- It seems adequate that the pre- and postsynaptic firing times t_0 and t_v, on which the weight change is based, are not too far from each other. On the other hand, our construction relies on the fact that the pair of pre- and postsynaptic spikes at t_u and t_v does not result in a change. Hence they should not be too close to each other. For biological systems, where learning relies on such time differences, the weight change approaches rapidly zero for larger time differences (see e.g. [Gerstner et al., 1996]). Hence an approximate fulfillment of our assumption should be sufficient.

- It must be guaranteed that the firing of v caused by u takes place while the EPSP is still in its initial linear segment. By raising the resting potential P_v^{rest} (e.g. by sufficiently many auxiliary EPSPs) and by restricting the range of possible weight values this requirement can be met easily.

The type of learning we have considered in this section is based on three spikes during each learning cycle, two presynaptic and one postsynaptic. The pair of presynaptic spikes is used to describe the input value, whereas one presynaptic and the postsynaptic spike define the synaptic modification. In the context of supervised learning this seems to be the minimal number of spikes required for a Hebbian-type learning based on temporal coding.

6.2 Parallel Learning

With the monosynaptic learning rule described in the previous section it is possible to learn single weights. However, usually a neuron v receives its input from several neurons $u_1, \dots, u_n$. The corresponding weights $w_{u_1,v}, \dots, w_{u_n,v}$ can still be learned in the way described above, if the weights are learned sequentially, such that v receives only inputs through one synapse during one learning cycle. Hence the question arises, how several weights can be learned in parallel more efficiently. If one would try to learn weights in parallel in a similar fashion as described in the previous section, the problem would arise of how to distribute the error encoded by $t_0 - t_v$ among the weights. Additional error signals would make Hebbian learning, where the weight change is only influenced by the time difference between some pre- and postsynaptic firing time, impossible.

A straightforward solution is possible if one allows a normalization of all weights of the neuron after each learning cycle. Such a weight normalization is frequently established in connection with the Hebb-rule since it prohibits the unlimited increase of weights which occurs otherwise (see e.g. [Brown and Chattarji, 1995]). Hence if one normalizes the weight vector $\underline{w} = (w_{u_1,v}, \dots, w_{u_n,v})$ such that $||\mathbf{w}|| = 1$ then a simple Hebbian learning rule can be formulated: the goal is that $\mathbf{w} = \tilde{\mathbf{w}}$ for some desired weight vector $\tilde{\mathbf{w}}$ with $||\tilde{\mathbf{w}}|| = 1$. In contrast to Section 6.1 v receives here through some additional synapse with a sufficiently strong weight an input spike such that v fires at time t_0. The i-th synapse $\langle u_i, v\rangle$ receives a spike from neuron u_i at time t_{u_i} such that $t_0 - t_{u_i} = \tilde{w}_{u_i,v}$. We ensure that v does not fire before t_0 by providing sufficiently many IPSPs through additional inhibitory synapses. One gets the following, two-step learning rule

$$\begin{array}{ll} (1) & \Delta w_{u_i,v} = \eta(t_0 - t_{u_i})\,, \qquad 1 \le i \le n \\ (2) & \text{normalize the resulting weight vector } \mathbf{w} \end{array} \tag{6.6}$$

where $\eta > 0$ is the learning rate, that has the following convergence property:

Theorem 6.2 *Given a neuron v with excitatory synapses $\langle u_i, v\rangle$ for $1 \le i \le n$, let $\tilde{\mathbf{w}}$ be a weight vector satisfying $||\tilde{\mathbf{w}}|| = 1$ and let $\eta > 0$. Then the sequence of weight vectors generated by the parallel learning rule (6.6) converges to $\tilde{\mathbf{w}}$ for any arbitrary initial weight vector.*

PROOF Let $\mathbf{w}^{(0)}$ be the initial vector and consider $\alpha_m = \mathbf{w}^{(m)} \cdot \tilde{\mathbf{w}}$ for $m \ge 1$. Due to the Cauchy-Schwarz inequality we get immediately $\alpha_m \le ||\mathbf{w}^{(m)}|| \cdot ||\tilde{\mathbf{w}}|| = 1$. We will show that $\alpha_{m+1}^2 > \alpha_m^2$ from which the convergence follows. Since $\tilde{\mathbf{w}} \cdot \tilde{\mathbf{w}} = 1$ we get from our learning rule (6.6) that

$$\alpha_{m+1}^2 \quad = \quad \left(\frac{(\mathbf{w}^{(m)} + \eta\tilde{\mathbf{w}}) \cdot \tilde{\mathbf{w}}}{||\mathbf{w}^{(m)} + \eta\tilde{\mathbf{w}}||} \right)^2$$

$$
\begin{aligned}
&= \frac{(\alpha_m + \eta\widetilde{\mathbf{w}}\cdot\widetilde{\mathbf{w}})^2}{(\mathbf{w}^m + \eta\widetilde{\mathbf{w}})^2} \\
&= \frac{(\alpha_m + \eta)^2}{1 + 2\eta\alpha_m + \eta^2}
\end{aligned}
$$

For $\mathbf{w}^{(m)} \neq \tilde{\mathbf{w}}$ we have $\alpha_m < 1$ which implies

$$(\alpha_m + \eta)^2 > (1 + 2\eta\alpha_m + \eta^2)\alpha_m^2.$$

It follows immediately that

$$\alpha_m^2 < \frac{(\alpha_m + \eta)^2}{1 + 2\eta\alpha_m + \eta^2} = \alpha_{m+1}^2$$

which completes the proof. ■

It is easy to verify that by some proper transformation of the normalization not only weights from $[0, 1]$ but from some arbitrary interval $[w_{\min}, w_{\max}] \subseteq \mathbf{R}^+$ can be learned.

In contrast to the monosynaptic rule this type of learning is not based on an error signal. Furthermore, we do not require here that EPSPs have a linear segment.

In the model of a spiking neuron we have explicitly excluded the presence of noise. However, our constructions in Sections 6.1 and 6.2 should also work in the case of a certain amount of noise, but they are harder to analyze theoretically. We expect that similar results can be shown using standard methods dealing with noise.

7 Competitive Learning with SNNs

In the following two chapters we will investigate possibilities for unsupervised learning with SNNs, where the network is required in the absence of any teacher to extract some information about the training data. This chapter examines how competitive learning can be performed by SNNs using temporal coding on the basis of the Maass-construction described in Chapter 5. We extend this idea to a learning mechanism that is closely related to one of the most successful paradigms of unsupervised learning: the self-organizing map (SOM) by Kohonen [Kohonen, 1995].

There are various regions in the brain which are known to be a topographic representation of some sensory surface. Some of the main qualitative features of these maps could be reproduced by the SOM (see e.g. [Ritter, 1995; Ritter et al., 1992]). For example, feature maps in area V1 of the visual cortex, including retinal position, orientation and ocular dominance, could be remarkably well reproduced by the SOM, even including many quantitative aspects [Blasel and Obermayer, 1994]. Hence the SOM provides a possible explanation how such maps can develop.

In previous versions of the SOM the input/output of a neuron was given by its firing rate and not by the timing of single firing events. Using lateral connections the procedure for detecting the so-called winner neuron is usually implemented as a recurrent network. This has the consequence that the winner neuron is detected not before the network has settled down into an equilibrium state. However, since the computation relies on the convergence of the network, this approach disregards the benefits that temporal coding offers to fast information processing in SNNs.

In addition to these conventional implementations there has also been some research on biologically more realistic models of self-organizing map algorithms, e.g. in [Kohonen, 1993; Sirosh and Miikkulainen, 1997; Choe and Miikkulainen, 1996]. Also in these approaches the output of a neuron is assumed to correspond to its firing rate, and learning takes place in terms of this rate after the network has reached a stable state of firing. In [Kohonen, 1993] it is suggested to implement the neighborhood-relations between biological neurons through varying extracellular concentrations of the molecule NO which has, however, the disadvantage that the diffusion and change in concentration of NO are rather slow (see also [Krekelberg and Taylor, 1997]).

This chapter proposes a mechanism for unsupervised learning in SNNs where computing and learning are based on the timing of single firing events. In contrast to the standard formulation of the SOM, our construction has the additional advantage that the winner among the competing neurons can be determined fast and locally by using lateral excitation and inhibition. These lateral connections also constitute the neighborhood relationship among the neurons. We assume that initially neurons which are topologically close together have strong excitatory lateral connections whereas remote neurons have strong inhibitory connections. During the learning process the lateral weights are decreased, thus reducing the size of the neighborhood.

In a series of computer simulations we have investigated the capability of the model to form topology preserving mappings. For the evaluation of these mappings, instead of relying on visual inspection, we used a measure for quantifying the neighborhood preservation which was recently studied [Goodhill and Sejnowski, 1997]. Our results show that the model exhibits the same characteristic behavior as the SOM. The typical emergence of topology preserving behavior could be observed for a wide range of parameters.

We also studied the effect of weight normalization. It is used in several implementations of the SOM (see e.g. [Choe and Miikkulainen, 1996; Kohonen, 1995; Sirosh and Miikkulainen, 1997]) but its biological relevance is controversial. We performed simulations with and without weight normalization after each learning cycle. Comparing the results we discovered that after a certain number of learning cycles approximately the same degree of topology preservation could be achieved regardless whether the weights were normalized or not. The model of unsupervised learning that we propose in this paper is therefore a candidate for a more realistic description of fast analog computation in biological neural systems.

Excerpts of this chapter were presented in [Ruf and Schmitt, 1998b, 1997c, 1998a].

7.1 Competitive Learning

On the basis of the Maass-construction we can realize competitive learning with SNNs as follows: given a set S of m-dimensional input vectors $\mathbf{s}^l = (s_1^l, \ldots, s_m^l) \in [0, \gamma]^m$ and an SNN with $m+1$ input neurons and n competitive neurons, where each competitive neuron v_j receives synaptic "feedforward" input from each input neuron u_i with weight w_{ij} and "lateral" synaptic input from each competitive neuron $v_k, k \neq j$, with weight $\tilde{w}_{kj}$. At each cycle of the learning procedure one $\mathbf{s}^l \in S$ is randomly chosen and the input neurons are made fire within the coding interval such that they temporally encode $\mathbf{s}^l$. As described in Section 5.1 we use an additional neuron u_0 which fires at time T_{in} encoding the value 0. Its weights are again chosen such that $\sum_{i=0}^{m} w_{ij} = \lambda$ for $1 \leq j \leq n$.

Each competitive neuron then starts to compute $\sum_i w_{ij} s_i^l$ as described in Section 5.1. However, only the firing time of the competitive neuron firing first actually expresses the corresponding weighted sum, the firing times of the other competitive neurons are

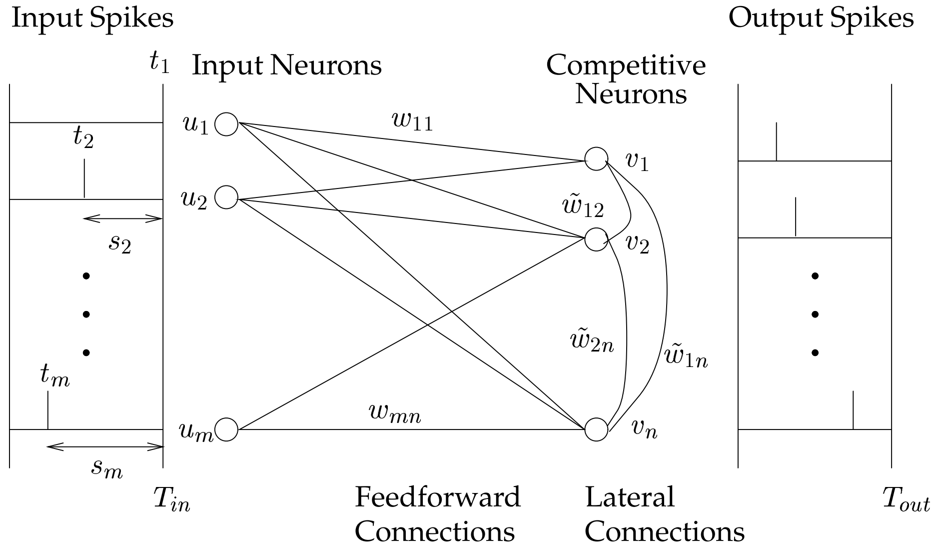

Figure 7.1: The basic architecture.

shifted by the firing of that neuron. If we assume now that the input vector and the weight vector for each neuron are normalized, then this weighted sum represents the similarity between the two vectors with respect to the Euclidean distance. Hence the earlier v_j fires, the more similar is its weight vector to the input vector, i.e. the winner among the layer of competitive neurons fires first (see Figure 7.1).

If the lateral connections are strongly inhibitory, such that the firing of the winner neuron, say v_k, inhibits all other neurons in the competitive layer from firing, we can implement in a straightforward way competitive learning: the standard competitive learning rule (also known as instar learning rule) simply has to be applied to the winner neuron v_k. This rule is given for v_k by

$$\Delta w_{ik} = \eta(s_i^l - w_{ik}), \qquad i \in \{1, \ldots, m\} \tag{7.1}$$

where $\mathbf{s}^l = (s_1^l, \ldots, s_m^l)$ is the current pattern and $\eta > 0$ the learning rate. Hence one learning cycle consists of the presentation of an input pattern $\mathbf{s}^l \in S$ making one of the competitive neurons fire. The spikes representing this pattern cause weight changes at the synapses of this winner neuron due to the learning rule (7.1).

This rule does not exclude explicitly that a weight may become negative. A weight vector of a neuron which has specialized on some input vectors will, however, contain only positive weights. A change of a weight sign can in addition be formally excluded by introducing a saturation function for the weights which guarantees that the weights assume values only from some bounded interval (see also Section 8.5).

7.2 Self-Organization

In order to extend this formulation of competitive learning to a realization of self-organization, one has to find a way to implement on the basis of locally available information a given neighborhood matrix $(m_{kj})_{1\leq k,j\leq n}$, where m_{kj} describes the distance between the kth and jth competitive neuron. We use a monotonously increasing function $m_{kj} \mapsto \tilde{w}_{kj}$ such that the lateral connections $\tilde{w}_{kj}$ among the competitive neurons reflect the structure of the neighborhood: initially neurons which are topologically close together have strong excitatory lateral connections whereas remote neurons have strong inhibitory connections. This means that the firing of the winner neuron, say v_k, at time t_k drives the firing times of neurons in the neighborhood of v_k towards t_k, thus increasing the values they encode. The firing of remote neurons is postponed by the lateral inhibition. In the example of Figure 7.1, the firing of the winner v_1 shifts the firing time of a topologically close neuron v_2 (which would compute a similar weighted sum if its firing were not influenced by v_1) towards the firing time of v_1 and makes a topologically remote neuron v_n fire later. We suggest the following learning rule:

$$\Delta w_{ij} = \eta \frac{T_{out} - t_j}{T_{out}} (s_i^l - w_{ij}) \tag{7.2}$$

where t_j is the firing time of the jth competitive neuron. The learning rule applies only to neurons that have fired before a certain time T_{out}. The factor $(T_{out} - t_j)/T_{out}$ realizes the neighborhood function, which is largest for the winner neuron and decreases for neurons which fire at later times.

During the learning process the lateral weights $\tilde{w}_{kj}$ are decreased, thus reducing the size of the neighborhood.[1] As in the standard formulation of the SOM, η is slowly decreased during learning.

7.3 Simulations

We tested our approach with one- and two-dimensional input patterns. This section summarizes the main results, technical details can be found in Section 11.1.

In the one-dimensional case 10 input patterns, uniformly distributed over the coding interval, were presented to a layer of 10 competitive neurons, which had initially random weights of values around the midpoint of the patterns. The lateral weights for the immediate neighbors were chosen slightly positive, for the second neighbors zero and for all other neurons negative. The goal was the formation of a linear map. Figure 7.2 shows that initially nearly all neurons react strongly, i.e. fire early on each in-

[1] This requires that the lateral weights change their sign during learning, which is not very realistic in the context of biological networks. However, one can consider instead two connections, one excitatory, which decreases and one inhibitory, which increases during learning.

put, whereas after learning a linear map has evolved such that only few neurons, being topologically close, react on a certain input pattern.

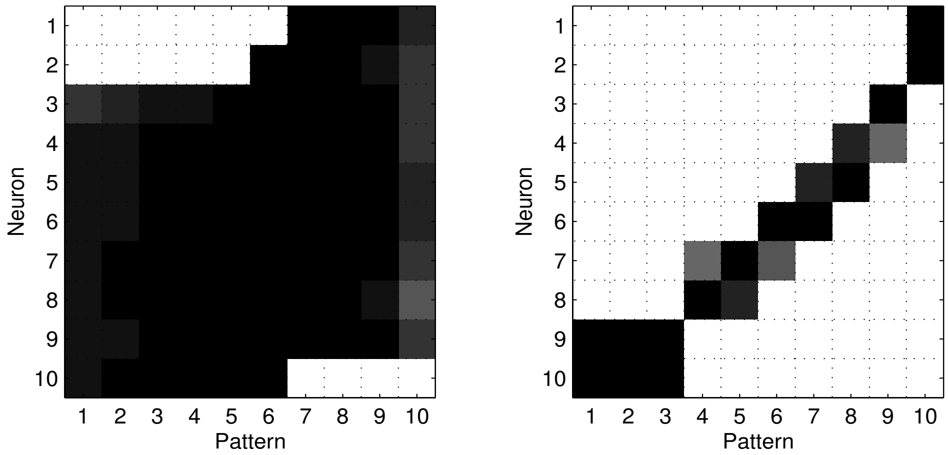

Figure 7.2: Learning process for one-dimensional input patterns after 10 iterations (left) and 4000 iterations (right). Each column represents the reaction of the network to a particular input pattern. The colors of the squares indicate the firing times: the darker a square, the earlier the corresponding firing time. The non-firing of a neuron is represented by a white square.

Furthermore, we investigated the behavior of our construction on one of the standard examples for the SOM, where two-dimensional input patterns are chosen randomly from a square region in the plane and the competitive neurons are expected to organize themselves in a topology preserving grid. For the tests we used an array of 5×5 competitive neurons. We normalized the inputs by adding a third input component, which was chosen such that all input vectors have the same norm. The weights were initialized in the same fashion as in the first experiment. In order to examine the effect of normalization we performed two series of experiments. In the first one the feedforward weights were normalized after each application of learning rule (7.2), in the second they remained unnormalized. Figure 7.3 shows the result of a simulation with weight normalization.

The topology preserving quality of self-organizing maps is usually assessed by visual inspection. This may be appropriate for a single mapping but it is hardly possible to compare the degree of topology preservation of two different mappings. In our second experiment we therefore used a measure for quantifying the neighborhood preservation known as "metric multidimensional scaling" (see e.g. [Goodhill and Sejnowski, 1997]). It is based on the objective function

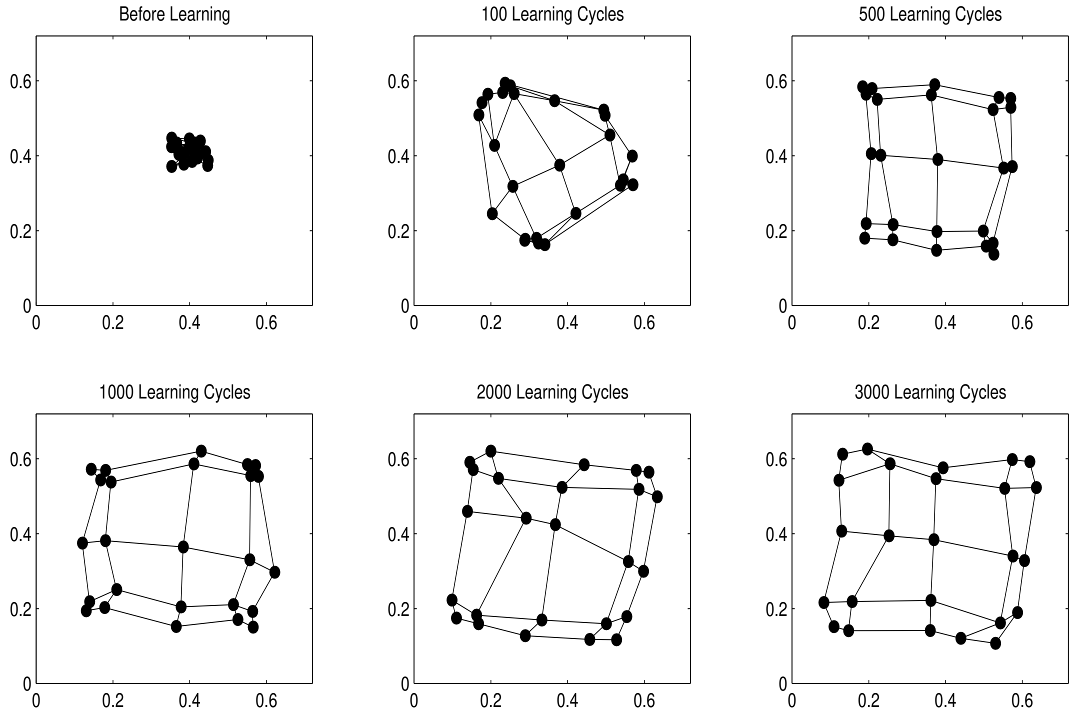

Figure 7.3: Learning a square region of the plane with 5×5 competitive neurons as described in the text. 3000 learning cycles were performed; each dot in a plot indicates the weight vector of a competitive neuron. Neighboring dots are connected by a line.

$$E_{\mathrm{MDS}} = \sum_{i=1}^{N} \sum_{j<i} (F(i,j) - G(M(i), M(j)))^2. \tag{7.3}$$

where N is the number of input patterns and M denotes the mapping of the network, i.e. $M(i)$ is the index of the winner neuron in the competitive layer for input $\mathbf{s}^i$. The matrix F represents the dissimilarity of a pair of input patterns $\mathbf{s}^i, \mathbf{s}^j$ expressed here by the Euclidean distance. The neighborhood matrix which defines the distances between pairs of competitive neurons is denoted by G. We used the familiar rectangular grid in two dimensions with the Manhattan metric for the neighborhood relationships in this example. Obviously, an optimal topology preserving mapping minimizes the value of E_{MDS}.

In order to scale the values of E_{MDS} into the interval $[0, 1]$ and to make the results for different initializations comparable, we defined the *relative neighborhood distortion* which is the actual value of E_{MDS} divided by the maximum initial value of E_{MDS}. The

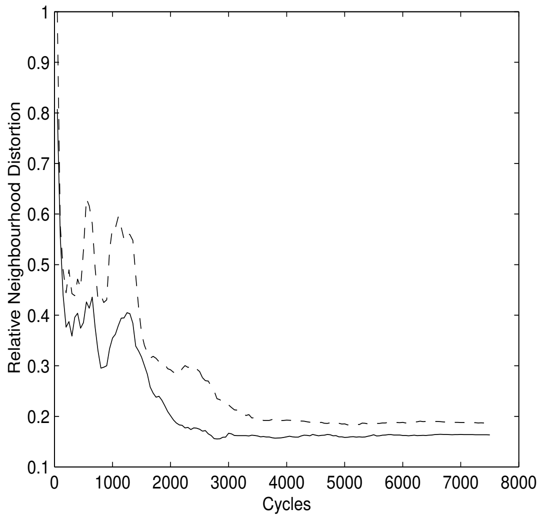

Figure 7.4: Development of the relative neighborhood distortion for the second example described in Section 7.3, where the solid (dashed) line shows the case with (without) weight normalization after each cycle.

results for two typical simulations are shown in Figure 7.4. Regardless whether the weights are normalized or not – the relative E_{MDS} stabilizes indeed at approximately the same small value.

A standard implementation of the SOM converges up to 10 times faster. This is due to the usage of a look-up table providing precise values for the neighbourhood function and also due to various heuristics e.g. for the decrementation of the learning rate and the reduction of the neighbourhood size, However, for this standard implementation E_{MDS} shows a qualitatively similar behaviour like ours as can be seen from Figure 7.5.

As mentioned above we have assumed that the neurons are of the leaky integrate-and-fire type, where the initial segment of the postsynaptic potentials rises linearly. In Chapter 9 we will address the question whether these assumptions can be dropped.

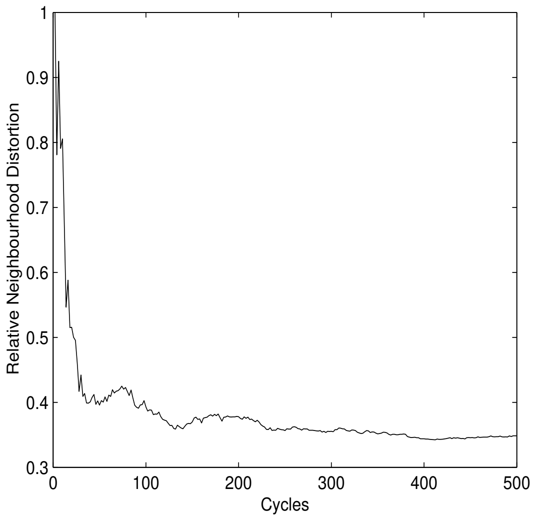

Figure 7.5: Same as Figure 7.4 but for a standard implementation of the SOM

8 Radial Basis Functions with SNNs

Among the most powerful ANN types are radial basis functions (RBFs), which have been successfully applied for example in the area of function approximation, pattern classification and data clustering (see [Haykin, 1994] for an overview). There is also growing interest in the relevance of this approach to biological neural networks. The question if biological neurons can realize one of the main advantages of RBF neurons, namely their ability to discover clusters in the input space is not yet resolved. In this chapter we present a learning algorithm for spiking neurons realizing RBFs which is not based on rate coding but on the timing of single spikes and which is able to find centers of clusters in an unsupervised fashion.

Hopfield presented a model for computing RBFs with spiking neurons [Hopfield, 1995] on a similar type of temporal coding as considered in the last chapters, but where no reference spike is necessary. The basic idea is that an "RBF neuron" encodes a particular input spike pattern in the delays available across its synapses. If an input pattern is close to the encoded spike pattern of an RBF neuron (called center of the RBF neuron in the following), the delays even out the differences of the firing times of the input neurons such that the RBF neuron fires. This approach is motivated by many neurobiological findings, e.g. in [O'Keefe and Reece, 1993] evidence has been given that the relative timing of spikes in the rat hippocampus might carry information about the relative location in an environment which is common to the rat. In [Haberly, 1985] it was shown that in the rat piriform cortex a wide range of time delays (up to 20 ms) can occur, which would allow a large interval for the input firing times.

In [Gerstner et al., 1996] a learning mechanism was introduced for a *single* RBF neuron. The proper delays, which encode the center of an RBF neuron, are chosen by using a neuronal learning rule which increases the efficacy of synapses with corresponding proper delays and decreases the efficacy of other synapses. A basic assumption underlying this approach is that there are several paths with different delays between each input and each RBF neuron.

Here we extend this approach by considering not only the firing/non-firing of a neuron but also its firing *time*. We show how on the basis of these ideas networks of such RBF neurons can be constructed and trained to divide the input space into several clusters, where the selection of the proper delays uses exclusively local information (essentially the difference between pre- and postsynaptic firing time). We performed computer simulations with neurons which were basically of the leaky integrate-and-fire type and achieved extremely promising results:

- The RBF neurons converged very reliably to the centers of the clusters even in the presence of noise.
- The RBF neurons were able to reconfigure themselves dynamically, if during the learning process clusters were added or removed. This adaptation to a changing environment seems to be of particular importance for biological systems.
- In the case where each cluster is formed in a certain subspace of the input space we show that with our learning mechanism RBF neurons can find the proper delays for the coordinates of the subspace and "deactivate" the remaining inputs. This can be considered as some kind of feature extraction, where the coordinates of the subspace carry the relevant information.
- Especially in the context of biological neural networks temporal sequences and their analysis seem to be of great importance. It turns out that by employing postsynaptic potentials of variable width such RBF neurons can detect temporal sequences even if those sequences are distorted in various ways.

Parts of this work appeared in [Natschläger and Ruf, 1997a,b].

8.1 A computational model for radial basis functions using spiking neurons

The underlying architecture of the SNNs considered here is very similar to the one presented in Chapter 7 for realizing a self-organizing behavior. The SNN consists of m input neurons $u_1, \ldots, u_m$ and n output neurons $v_1, \ldots v_n$ (see Figure 8.1). To the output neurons we will refer to as RBF neurons in the following. In the simplest case each input neuron u_i forms one synaptic connection to each RBF neuron v_j with weight w_{ij} and delay d_{ij}. The center of an RBF neuron v_j is given by the vector $\mathbf{c}_j = \langle c_{j,1}, \ldots, c_{j,m} \rangle$ with $c_{j,i} = d_{ij} - \min\{d_{ij} | 1 \le i \le m\}$.

Similar as in the Maass-construction we consider the simple coding scheme, where each input neuron u_i fires exactly once at time t_i during the coding interval $[T_{in} - \tilde{\gamma}, T_{in}]$ with $\tilde{\gamma}$ being a constant to be determined later. Here, the input is encoded relative to the first spike in the coding interval, i.e. there is no explicit reference spike necessary. We will denote an input vector in this type of coding with $\mathbf{x}$. It is given by $\mathbf{x} = \langle x_1, \ldots, x_m \rangle$ with $x_i = \max\{t_i | 1 \le i \le m\} - t_i$.

The input $\mathbf{x}$ is close to the center $\mathbf{c}_j$ of an RBF neuron v_j if the spikes of the input neurons reach the soma of v_j due to the corresponding delays at similar times, i.e. if $\|\mathbf{x} - \mathbf{c}_j\|$ is small. This is basically the approach presented in [Hopfield, 1995]. He considers

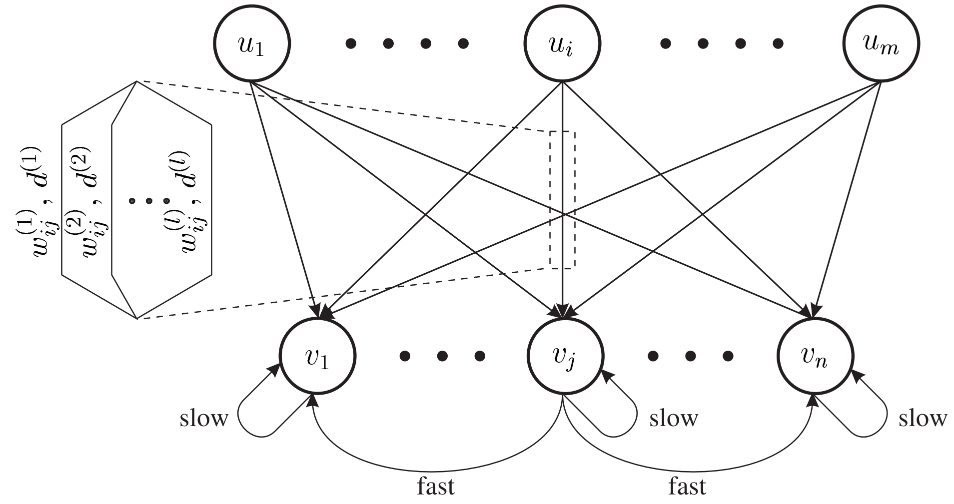

Figure 8.1: Our basic architecture for an RBF network of spiking neurons. Each connection from an input neuron u_i to an RBF neuron v_j consists actually of a set of l excitatory synapses with different weights $w_{ij}^{(k)}$ and corresponding delays $d^{(k)}$, as indicated by the magnification on the left. Each RBF neuron forms a slow inhibitory connection to itself and fast inhibitory connections to all other RBF neurons.

the case, where the input vector is close enough to the center of an RBF neuron to make v_j fire. However, in addition the firing time of v_j indicates, how close an input vector $\mathbf{x}$ is to $\mathbf{c}_j$ (see Figure 8.2). If the distance between $\mathbf{x}$ and v_j is too large, v_j does not fire at all. Let us consider now a set of RBF neurons $\{v_1, \ldots, v_n\}$. If for some input vector $\mathbf{x}$ the difference $\|\mathbf{x} - \mathbf{c}_j\|$ is small enough for various j to make v_j fire then the RBF neuron whose center is closest to $\mathbf{x}$ fires first.[1] Hence a set of such RBF neurons can be used to separate inputs into various clusters.

[1]Strictly speaking, there are some cases where a small deviation in the input spikes from the center of an RBF neuron v_j may cause an *earlier* firing. Consider e.g. the case where the input vector equals the center of v_j except one coordinate i, such that neuron u_i fires slightly earlier. This makes v_j fire earlier if the EPSP caused by u_i is still in its rising segment when v_j fires. However, since we choose a threshold being high enough to make v_j only fire when it received synchronous spikes from nearly all input neurons, v_j tends to fire more towards the end of the rising segment of the EPSPs. This shift in the firing time of v_j requires very special constellations of input spikes, which are especially for large numbers of inputs very unlikely to occur. Hence this undesired effect can be neglected.

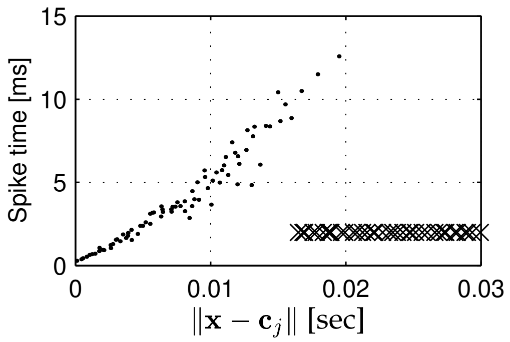

Figure 8.2: Dependence of the firing time of an RBF neuron v_j on the distance of the input vector $\mathbf{x}$ to the center $\mathbf{c}_j$. For this simulation 120 uniformly distributed inputs $\mathbf{x} \in [0, 20\,\mathrm{ms}]^{20}$ are presented to v_j, with equal weights and delays uniformly distributed over $[0, 20\,\mathrm{ms}]$. Crosses indicate the case that the RBF neuron has not fired.

8.2 Learning RBFs and Clustering with spiking neurons

We have seen now that RBF neurons can detect clusters by making use of their firing times. Now we show that the centers of the clusters can be found by the RBF neurons in an unsupervised way: we assume similar as in [Gerstner et al., 1996] that each u_i forms to each v_j instead of one synapse a set of synapses with varying presynaptic delays (see Figure 8.1). Throughout this chapter we use the set $D = \{d^{(1)}, \ldots, d^{(l)}\}$ of delays which is the same for every input with $d^{(k)} - d^{(k-1)} > 0$ for $2 \leq k \leq l$. Hence the total delay $d_{ij}^{(l)}$ between the presynaptic firing and the time the postsynaptic potential starts rising at the soma results from the time $d^{(l)}$ the spike emitted from u_i needs to activate the synapse $\langle u_i, v_j \rangle$ corresponding to this delay plus a time d^{post}, which is assumed to be constant here, needed until the potential at the soma of v_j starts rising due to the resulting PSP. The length $\tilde{\gamma}$ of the coding interval where the input neurons are allowed to fire is assumed to be smaller than $d^{(l)} - d^{(1)}$. We say the delay of length $d^{(k)}$ for the connection from u_i to v_j is *active* if its corresponding weight $w_{ij}^{(k)}$ is significantly greater than zero.

The goal of our learning algorithm can be formulated as follows: given a cluster C of input vectors $\mathbf{x}$ in the input space, an RBF neuron v_j should activate for each input coordinate one delay, such that the resulting center $\mathbf{c}_j$ minimizes $\sum_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{c}_j\|$. In addition we also want to allow that several neighboring delays become activated because the center of a cluster may not be representable due to a finite number of available delays. As it will turn out (see Section 8.2), the activation of a larger number of neighboring delays may also reflect a larger variance of the input cluster.

For such a constellation of several active neighboring delays we define the center $\mathbf{c}_j$ of an RBF neuron v_j as follows: we compute for every input coordinate i the mean delay

$$\tilde{d}_{ij} = \sum_{k=1}^{l} \frac{w_{ij}^{(k)} \cdot d^{(k)}}{\sum_{k=1}^{l} w_{ij}^{(k)}}$$

The center of v_j is then the vector $\mathbf{c}_j = \langle c_{j,1}, \ldots, c_{j,m} \rangle$ with $c_{j,i} = \tilde{d}_{ij} - \min\{\tilde{d}_{ij} | 1 \leq i \leq m\}$.

A convergence of each RBF neuron to the center of some cluster can be achieved in the following way: initially all weights are set to random values of a magnitude such that no RBF neuron can fire until it has received at least one spike from every $u_i, 1 \leq i \leq m$ within the coding interval. As soon as an RBF neuron fires, the spike is propagated backwards to its synapses, where a weight is changed according to some "learning-function" $L(\Delta t)$, with $\Delta t = t_{pre} - t_{post}$ denoting the difference between the arrival times t_{pre} and t_{post} of the pre- and postsynaptic spike at the synapse (see Figure 8.3). This is basically the learning mechanism suggested in [Markram and Sakmann, 1995; Markram et al., 1997]. In the following, we will use the learning function as shown in Figure 8.4. $L(\Delta t)$ is chosen here such that the weights of those synapses, which received shortly before the postsynaptic firing a presynaptic spike, are maximally increased (i.e. the peak of L occurs at a value smaller 0, here at $\Delta t = 2\,ms$), whereas synapses, which received a presynaptic spike much earlier or later are decreased.

More precisely, the weight $w_{ij}^{(k)}$ of the synapse between input neuron u_i and RBF neuron v_j with delay $d^{(k)}$ is updated during the whole learning process due to the following learning rule:

$$\Delta w_{ij}^{(k)} = \eta L\left(t_{pre} - t_{post}\right) = \eta L\left((t_i + d^{(k)}) - (t_j + d^{back})\right) \tag{8.1}$$

for every pair of input firing time t_i and firing time t_j of the RBF neuron (see Figure 8.4 for an example). $\eta > 0$ denotes the learning rate. The delay d^{back} describes the time the postsynaptic spike needs to propagate backwards to the synapses. We assume that d^{back} has for all these synapses the same value, i.e. it is independent from the position of the synapse on the dendritic tree, which is supported by experimental findings as in [Stuart and Sakmann, 1994] where it is shown that it takes about $1\,ms$ for a spike generated at the soma to propagate to the most distal dendrites. Furthermore, we assume that the weights saturate at 0 respectively at a certain $w_{max} > 0$ (see Section 8.5 for details).

By adding fast lateral inhibition among the RBF neurons, one can implement a winner-take-all mechanism where only the RBF neuron fires which is closest to the current input vector. Hence, only the weights of the winning RBF neuron are modified, such that its center moves towards the current input vector. This is a similar approach as in competitive learning as described in Chapter 7 with the difference that the relevant information is stored here in the delays.

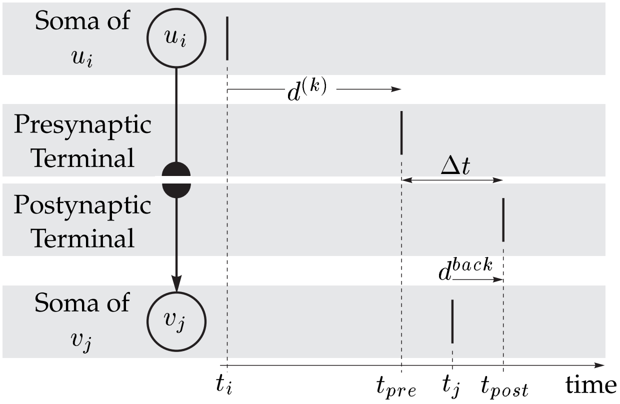

Figure 8.3: A spike emitted at u_i (v_j) at time t_i (t_j) reaches the synapse between u_i and v_j at time $t_{pre} = t_i + d^{(k)}$ ($t_{post} = t_j + d^{back}$).

As mentioned earlier we want that the firing *time* of an RBF neuron reflects how close its center is to the current input vector. If the learning rule had now increased the weight of one synapse without changing any other weight, this would make the neuron fire earlier when the same input vector is applied. Within this context this is an undesirable effect, since we want that the firing time of an RBF neuron is given by the peaks of the weight distribution representing the proper delays. Such peaks can be seen in Figure 8.5, where there are only a few neighboring delays with large corresponding weights after learning. I.e. the learning process should basically shift this distribution. Hence we have to make sure that the sum of the weights of all incoming synapses remains approximately constant while applying the learning rule (8.1). To achieve such a constant sum, one would usually have to require some non-local interaction. However, if one chooses a proper learning-function (where $\int_{-\infty}^{\infty} L(t)dt \approx 0$), this effect can be also achieved using only locally available information. Figure 8.6 shows that the peaks of the membrane potential caused by (similar) input vectors from one cluster indeed stabilize during the learning process at a certain level.

It is possible that some RBF neurons specialize on sets of clusters (thus computing their mean) and other RBF neurons are not used at all. We solved this problem, which is a general clustering problem, by adding slow self-inhibition to every RBF neuron, decreasing the probability that an RBF neuron fires twice within a few iterations of the learning rule. A similar "conscience" mechanism is often used in competitive learning [Grossberg, 1976]. However, the same effect can be achieved without self-inhibition in various other ways like adaption or with a surplus of RBF neurons choosing both the learning function and the initialization of the weights properly, such that each RBF neuron needs at least one spike from each input neuron.

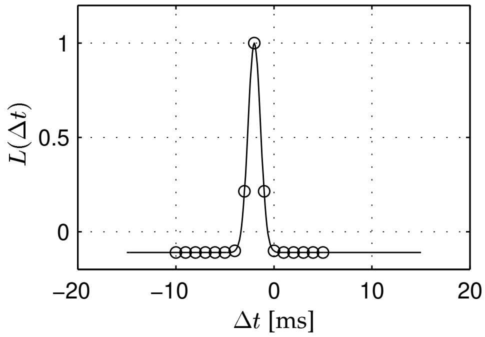

Figure 8.4: Learning function $L(\Delta t)$, describing how the weight change is influenced by the difference $\Delta t = t_{pre} - t_{post}$ between pre- and postsynaptic firing times at the synapse. The circles denote the weight changes for various delays occurring when u_i fires 10 ms before v_j. The leftmost (rightmost) circle corresponds to a delay of 1 ms (16 ms). We chose here $d^{back} = 1$ ms.

Simulation Results

We performed computer simulations in order to investigate how such RBF neurons behave on a high-dimensional input space. We considered the case where there were p vectors in the input space each forming the center of one cluster C_μ for $1 \leq \mu \leq p$. The input vectors were then generated by choosing one of the p centers (where each center was chosen with the same probability $1/p$) and then adding Gaussian noise to that center. In the following we use RBF neurons with 40 inputs. The length of the coding interval was 10 ms, the delay-interval 15 ms such that the available delays were $1\,\mathrm{ms}, 2\,\mathrm{ms}, \ldots, 16\,\mathrm{ms}$. Every 50 ms a new learning cycle was started by presenting a new input vector $\mathbf{x}$ to the network. For each component of an input vector $\mathbf{x} \in C_\mu$ the standard deviation was chosen between 2 ms and 4 ms. There was no overlap between the clusters. We set $L(\Delta t) = 0$ for $|\Delta t| > 15\,\mathrm{ms}$ to make sure that weights are changed only due to the current input vector.

- $n = p$: In the case where there were as many RBF neurons as clusters we could observe that after about 50 examples from each cluster each RBF neuron had converged to one cluster. As Figure 8.5 indicates, the variance of the clusters is reflected in the distribution of the weights of the delays. A small standard deviation of a cluster results in a sharp peak, whereas larger values cause several similar delays to have strong weights nearby the saturation value w_{max}, such that the RBF neuron will respond stronger, i.e. fires earlier, for a wider range of input vectors. Furthermore, we could observe that an RBF neuron fires considerably earlier after learning a cluster, when a pattern from that cluster is presented. This results from the fact that the membrane potential increases more steeply after learning

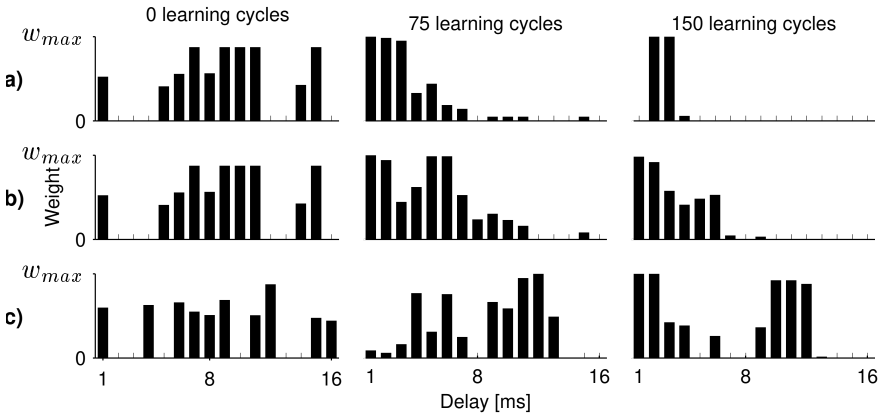

Figure 8.5: Development of the weights for a set of delays $D = \{1\,\mathrm{ms}, \ldots, 16\,\mathrm{ms}\}$ for one input neuron and one RBF neuron. a) Input patterns chosen from a cluster C_μ with small standard deviation. b) Same as (a), but with large standard deviation. c) Input patterns chosen from two different clusters.

(see Figure 8.7).
If one considers the definition of a center $\mathbf{c}_j$ given in Section 8.1, there can be several values of $\tilde{d}_{ij}$ which yield the same center $\mathbf{c}_j$. Observe that our model implies that $\min\{\tilde{d}_{ij}|1 \leq i \leq m\}$ tends to be equal to the smallest available delay $d^{(1)} = 1\,\mathrm{ms}$. This is not obvious since we use no reference spike to define our input interval. It has the advantage that the firing times of several RBF neurons are comparable on an absolute time scale as suggested in [Thorpe and Imbert, 1996].

- $n < p$: In the case of more clusters than RBF neurons, some RBF neurons will converge to the mean of a subset of the clusters. This may result in one wide peak or several smaller peaks in the distribution of the strengths of the delays, which is the best one can expect in this case (cf. Figure 8.5c). The latter case occurs if an RBF neuron has specialized on several clusters, i.e. it has more than one center.

- $n > p$: If there are initially more RBF neurons than clusters, it is obvious that several RBF neurons can specialize on the same cluster or that, depending on the initial conditions, some RBF neurons do not learn anything. Furthermore, we observed that a dynamic reconfiguration of the RBF neurons in a changing environment is possible. We assume that some of the RBF neurons have already learned to represent some clusters. If one adds during the learning process another cluster, one of the "free" RBF neurons will quickly specialize on this cluster (see Figure 8.8). If one continues adding clusters, such that the number of clusters exceeds the number of RBF neurons, a dynamic reconfiguration occurs in the sense, that an RBF neuron switches from an old to a new cluster or learns the

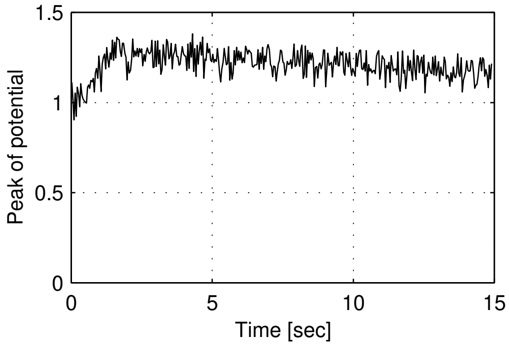

Figure 8.6: Development of the peaks of the membrane potential for an RBF neuron during the learning process. The jitter results from the varying input vectors chosen from one cluster. For the threshold we chose for both figures $\Theta = 1.0$.

mean of a set of clusters including new ones.

We also investigated the influence of noise by making each input neuron fire in addition at random time points. This means that each input neuron generates a Poisson spike train (of low frequency) and fires in addition corresponding to the sequence of input patterns. Figure 8.9 shows that despite of these underlying Poisson spike trains the RBF neurons are still able to detect the patterns in the noisy environment and assign them to the proper cluster. We were able to show that such noise may be even present during the learning phase without deteriorating the learning quality and speed considerably.

8.3 Finding Clusters in Subspaces

If one interprets the input coordinates as a representation of certain features, then the clusters represent typical constellations of frequently occurring combinations of input values. A cluster may then be formed out of some features, whereas other features are irrelevant. In our context, this means that only certain input neurons describe a cluster whereas the remaining input neurons simply produce noise. The task of the RBF neuron is to extract the "relevant" inputs for every cluster.

More precisely, we consider here an input space of dimension n with basis $E = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$, where $\mathbf{e}_i$ is the ith unit vector. Let us assume that for an input $\mathbf{x} = \sum_{i=1}^{n} \lambda_i \mathbf{e}_i$ the input neuron u_i receives the value $\lambda_i \in [0, \tilde{\gamma}]$, with $\tilde{\gamma}$ being the length of the coding interval. We consider only subspaces where the basis E_{sub} is a subset of

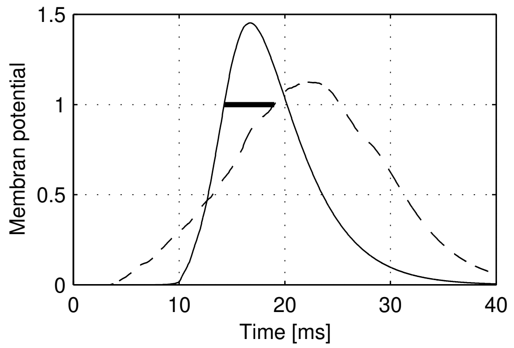

Figure 8.7: Membrane potential of an RBF neuron for one input vector before (dashed line) and after (solid line) learning, causing a shift of the firing time from 19 ms to 14 ms, denoted by the horizontal bar at the height of the threshold.

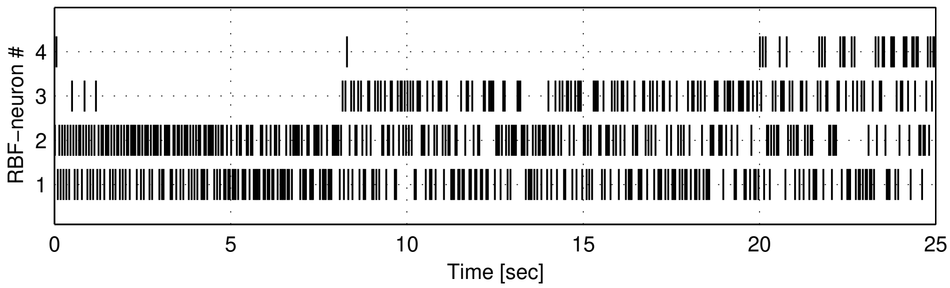

Figure 8.8: Spike trains for four RBF neurons. At the beginning, two clusters were presented, after 8 s a third and after 20 s a fourth cluster was added.

E. Samples from a cluster C in such a subspace are presented to the network as

$$\mathbf{x} = \sum_{\mathbf{e}_i \in E_{sub}} \lambda_i \mathbf{e}_i + \sum_{\mathbf{e}_i \in E \setminus E_{sub}} n_i \mathbf{e}_i$$

where the n_i are random variables, uniformly distributed over $[0, \tilde{\gamma}]$ (see Figure 8.10 for an example). Assume that RBF neuron v_j is to converge to the center $\mathbf{c}$ of C. The goal is that besides finding the proper delays for the input neurons receiving the inputs for E_{sub}, the remaining "noisy" coordinates should have no impact on v_j, i.e. the corresponding weights should become zero.

Our construction described in the previous section can handle this situation if one slightly changes the learning function, such that larger values of $|\Delta t|$ cause a larger weight decay. This results on the long run in a weight decay for all delays of these noisy coordinates and in sharp peaks of the weight distribution for the other inputs u_i

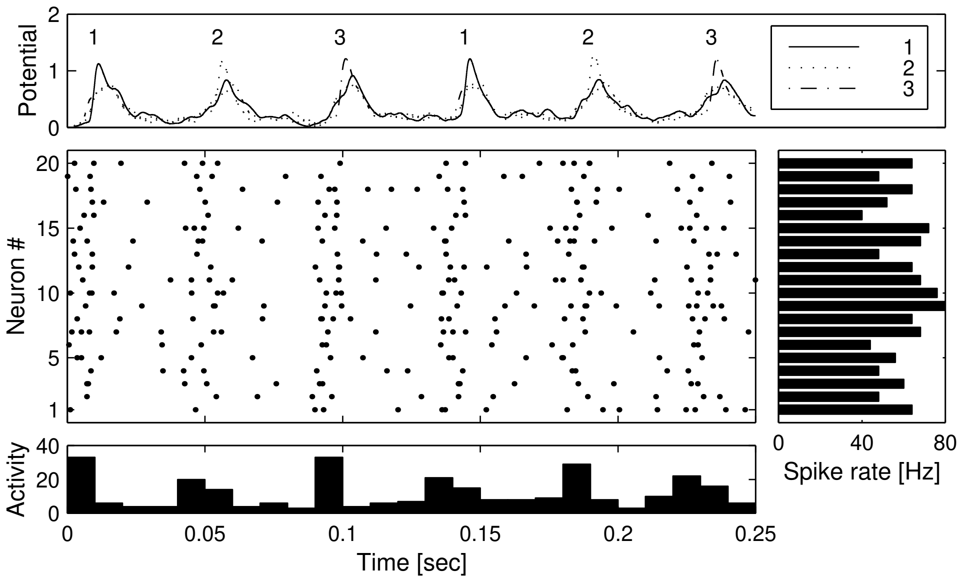

Figure 8.9: Behavior of 3 RBF neurons on spike trains including Poisson distributed spikes of 25 spikes/sec after 270 learning cycles. The vectors presented to the network (every 50 ms) are chosen cyclically from 3 clusters. The lower box shows the number of spikes within time bins of size 10 ms. Although the input neurons fire all approximately with the same frequency (right box) the RBF neurons can distinguish the pattern in the noise (upper box). The threshold was set to 1.0.

with $\mathbf{e}_i \in E_{sub}$. Figure 8.11 shows an example of a delay distribution after a successful learning process. It also indicates that RBF neurons which have finally not specialized on some cluster may have quite similar weight distributions as "successful" RBF neurons. This results from a rather long period of competition.

8.4 Temporal Sequence Recognition

Since the input patterns are temporally encoded in our approach, it is quite natural to use it for the recognition of regularities in temporal sequences of input stimuli. This is of particular importance in biological systems (e.g. in audition and vision) but also for related tasks in hardware implementations. We will show that it is possible to recognize sequences which are distorted in time and form.

Let us consider a spatio-temporal input firing pattern which has the form $P = \langle (i_1, t_1), \ldots, (i_N, t_N) \rangle$ with $T_{in} - \tilde{\gamma} \leq t_1 \leq t_2 \leq \ldots \leq t_N \leq T_{in}$, where (i_j, t_j) indicates

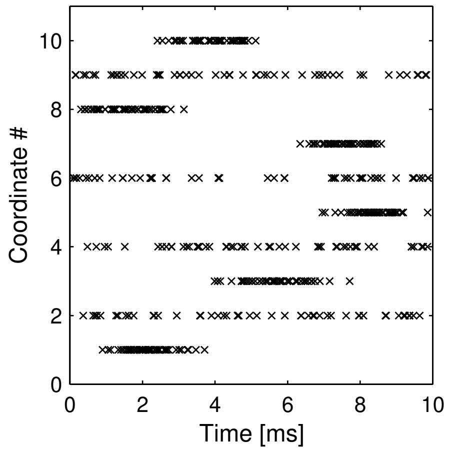

Figure 8.10: Distribution of 100 input vectors for a cluster of dimension 6 in the subspace $E_{sub} = \{\mathbf{e}_1, \mathbf{e}_3, \mathbf{e}_5, \mathbf{e}_7, \mathbf{e}_8, \mathbf{e}_{10}\}$. The coding interval is of length 10 ms.

that at time t_j input neuron u_{i_j} generated a spike.[2] We drop our previous assumption that an input neuron may fire only once within the coding interval, i.e. we allow $i_j = i_k$ for $j \neq k$. The detection of such patterns corresponds to the clustering problem described above, except that we allow here that an input neuron may fire more than once within one sequence. In Section 8.2 we generated the clusters by adding Gaussian noise to the centers of the clusters. Since we are now dealing with temporal sequences, we consider two types of noise, namely distortion of the sequences in form and in time: the form of a sequence may be distorted if an input spike at a certain time is produced by an erroneous input. A time warp can occur by stretching (squeezing) the sequence, i.e. the firing times are scaled such that the firings take place in a larger (smaller) time interval $K \cdot (t_N - t_1)$ for $K > 1$ ($K < 1$). Furthermore, it may be possible that "wrong" spikes appear in the sequence postponing the subsequent spikes of the sequence, i.e. a spike at time t of neuron u_{i_k} with $t_j \leq t \leq t_{j+1}$ would change P to the pattern $\tilde{P} = \langle (i_1, t_1), \ldots, (i_j, t_j), (i_k, t), (i_{j+1}, (t - t_j) + t_{j+1}), \ldots, (i_N, (t - t_j) + t_N) \rangle$. This type of noise is of particular importance for temporal sequences (e.g. sound waves like speech), if the "correct" sequence is interrupted by some noise and then continued.

An RBF neuron v, which has been tuned for such a P, can still fire for an input pattern $\tilde{P}$, if the EPSPs caused by the input spikes before t have a longer impact on the potential of v. This can be achieved, if the EPSPs of these earlier spikes last longer (see Figure 8.12), such that a shift of the later EPSPs of the amount $t - t_j$, caused by the input spikes after t, still suffices to make v fire. Generally we assume that the earlier an input

[2]We assume that P describes a valid spike train, where a neuron does not fire within its absolute refractory period.

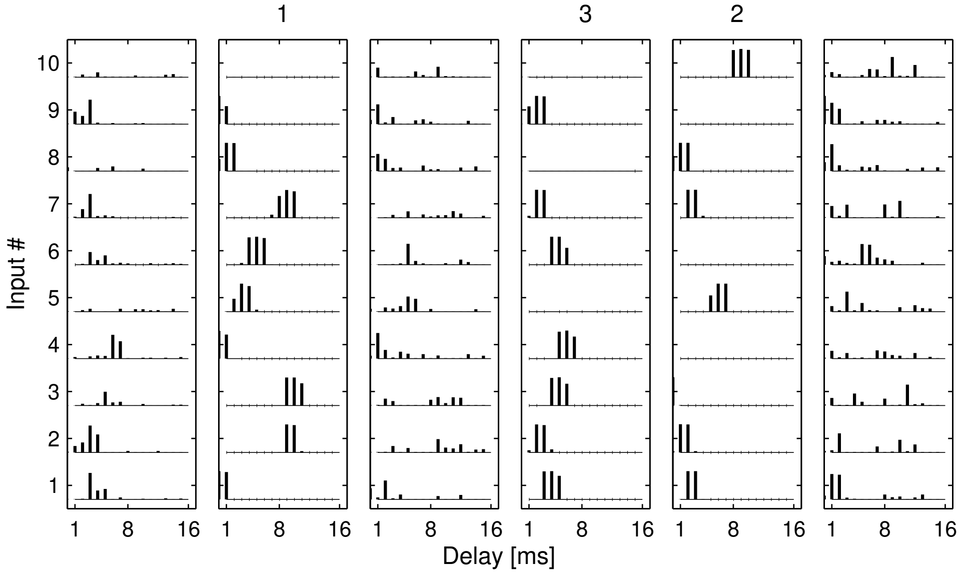

Figure 8.11: Weight distribution for a set of delays $D = \{1\,ms, \ldots, 16\,ms\}$ after a successful learning process of 660 learning cycles with three clusters. Each column represents one RBF neuron with 40 inputs, of which 10 are shown. A number above a column indicates that the corresponding RBF neuron has specialized on the cluster with that number. Cluster 1 had 5, cluster 2 had 10 and cluster 3 had 15 noisy coordinates. The first RBF neuron has a similar weight distribution as the fourth RBF neuron, since it has competed for a rather long period with this fourth neuron for cluster 3.

spike occurs the wider is the resulting EPSP.

This approach is similar to an idea presented in [Tank and Hopfield, 1987], where it is shown how an ANN can recognize such sequences using certain ”delay filters”. We were able to realize this concentration in time very naturally by spiking neurons employing EPSPs of variable width which roughly correspond to the delay filters.

Our construction is able to deal with all the above-mentioned types of noise. Figure 8.13 shows as an example that a stretching or squeezing of an input sequence up to a factor of $K = 1.4$ respectively $K = 0.5$ still allows a proper detection of each sequence. If a sequence is too strongly squeezed or stretched, the RBF neuron, which was tuned to the corresponding sequence, does not fire at all, i.e. the network does either respond correctly or does not respond at all.

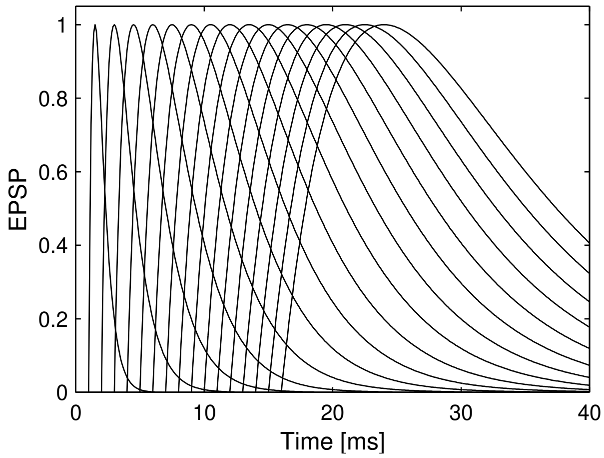

Figure 8.12: EPSPs used for our simulations described in Section 8.4. The EPSPs shown here were α-functions, (see Equation (9.2)), with $\tau_1 = \tau_2 \in \{0.5\,ms, 1.0\,ms, \ldots, 8\,ms\}$. The delays (i.e. the time until the onset of the EPSP) were chosen $2 \cdot \tau_1$ such that a wider EPSP corresponds to a larger delay.

8.5 Stability

It is not obvious that the RBF neurons stabilize at a certain delay constellation during a long learning process. Indeed, it could be the case that the center of an RBF neuron becomes smaller during the learning process such that for certain inputs only the smallest available delays are active (i.e. have weights greater zero) whereas for other inputs the active delays vanish (see Figure 8.14). This undesirable behavior occurs if some weights become too large during the learning process. Large weights make the neuron fire earlier, such that the learning rule (8.1) increases the weights corresponding to shorter delays. Our learning rule tends to increase weights until they have reached the saturation value w_{max} (i.e. the largest possible weight). Hence this shift of delays can be avoided if w_{max} is not too large. The opposite effect, where larger delays are activated, may occur if w_{max} is too small. Thus if the learning process lasts very long, a good choice of w_{max} is important for the stability (Section 11.2 describes how we chose w_{max} for our simulations.).

The influence of a proper choice for w_{max} on the stability is considerably reduced by using in addition a continuous saturation function $S(w)$ such that a weight which is close to saturation is less strongly modified than a weight being more in the middle of $[0, w_{max}]$ (see Figure 8.14c for an example). Hence, such a saturation function supports the stabilization of possible peaks in the weight distribution. One may use the following

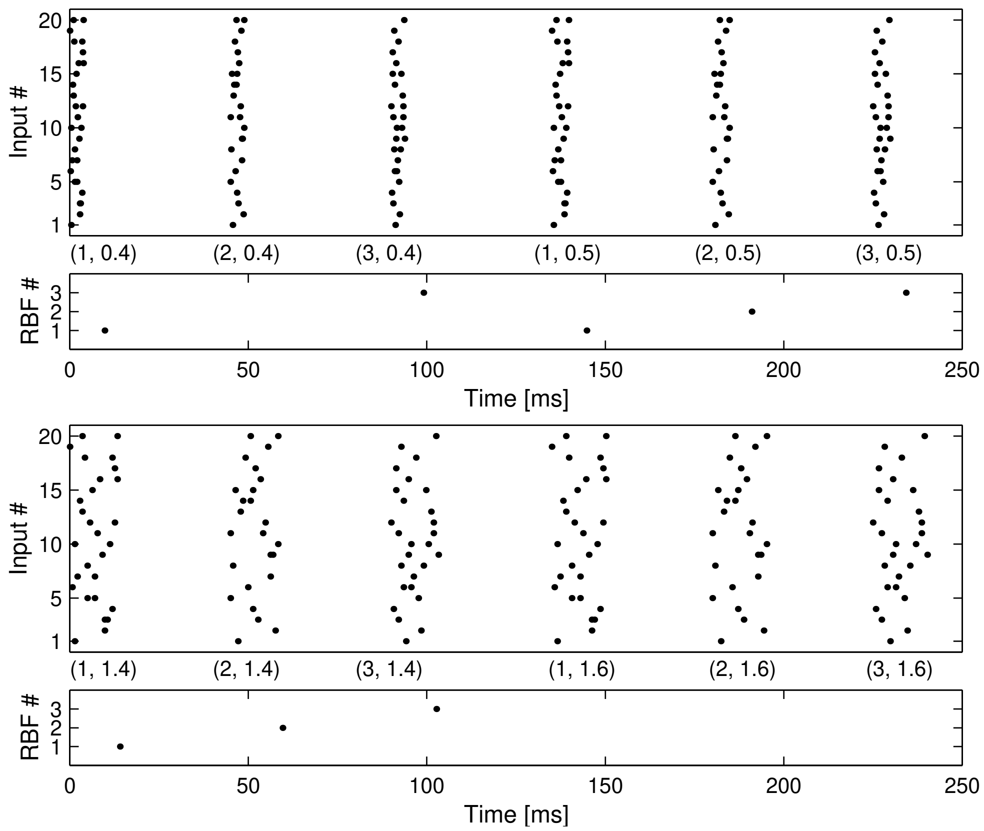

Figure 8.13: Input- and output spike trains of an RBF network with 20 inputs and 3 RBF neurons. The network was trained on three clusters. After 270 learning cycles RBF neuron v_j had converged to the center of cluster j. A tuple (j, K) indicates that an input vector from the jth cluster with a stretch/squeeze factor of K was presented.

modification of the learning rule (8.1):

$$w_{ij}^{(k),new} = w_{ij}^{(k),old} + \eta \cdot S(w_{ij}^{(k),old}) \cdot L((t_i + d^{(k)}) - (t_j + d^{back})) \tag{8.2}$$

If the weights are outside of the interval $[0, w_{max}]$ after applying this rule, an additional weight clipping is made, such that a weight can never exceed its boundaries. In order to allow dynamic reconfigurations (like adding/removing clusters during the learning process), one has to assume that $S(0) > 0$ and $S(w_{max}) > 0$.

Without a saturation function, a reasonable choice of w_{max} is important especially in the context of finding clusters in subspaces (Section 8.3): One basic assumption in our construction is that one spike of each input neuron within the coding interval suffices to make an RBF neuron fire. If there are some noisy coordinates, the corresponding

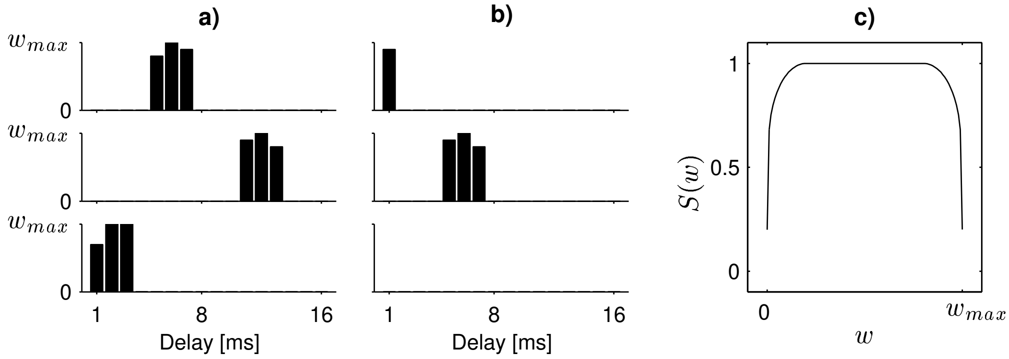

Figure 8.14: a) and b) Shift of the weight distribution for a set of delays $D = \{1\,\mathrm{ms}, \ldots, 16\,\mathrm{ms}\}$. The maximum weight w_{max} was chosen too large, such that an initial learning success after 120 learning cycles (a) shifted towards smaller delays, where for certain inputs even no delay had a weight greater zero (b). c) One possible choice for a saturation function.

input neurons have no impact on the RBF neuron after a successful learning process. Hence the weights for the remaining input neurons must be large enough to make the RBF neuron fire. The straightforward solution to this problem, namely to have various w_{max} for different RBF neurons depending on the number of noisy coordinates is not very desirable in the framework of unsupervised learning since one must put some knowledge about the training set into the parameters of the network. In our simulations we were able to show that it suffices to use a saturation function as shown in Figure 8.14c and one *global* value for w_{max}, which is large enough that an RBF neuron with many noisy coordinates can still fire but also can stabilize at a certain delay constellation.

8.6 Conclusions

We have presented a method for clustering temporally encoded input patterns with networks of spiking neurons. It can be extended in various ways, allowing the realization of feature extraction and the recognition of temporal sequences.

In biological neural systems there exist at least two different types of inhibition: fast $\mathrm{GABA_A}$ and slow $\mathrm{GABA_B}$ mediated inhibitory synapses (see [Segev, 1995] for an overview of time constants). Our model combines the two in a new way: we used fast lateral inhibition to implement the winner-take-all mechanism and slow self-inhibition to allow a competition among all RBF neurons, such that no RBF neuron dominates the competition over a longer period of time.

In RBF layers for ANNs one frequently considers an additional linear gate behind the RBF layer, which can be used for classification and function approximation tasks. If the inputs are temporally encoded as described in Section 8.1, spiking neurons can compute such linear gates using the Maass-construction described in Chapter 5. This idea can be integrated into our construction as follows: if one omits the fast lateral inhibition among the RBF neurons, the firing time of each RBF neuron relative to the others expresses its similarity to the input vector. A spiking neuron receiving input from all these RBF neurons can then compute the linear gate.

9 Biologically More Realistic Models of Spiking Neurons

In this chapter we focus on the question whether the basic principles of the Maass-construction can be still used for biologically more realistic models, where the EPSPs do not contain an initial linear segment and where one also takes nonlinear effects in the dendrites and Hodgkin-Huxley kinetics at the soma as described in Section 2.2 into account. With the help of the GENESIS simulation software, which will be described in brief in Section 9.1, we demonstrate in Section 9.2 that such a neuron can compute in a very natural way linear functions in temporal coding, only assuming that the neuron receives all its inputs within a certain time window. In Section 9.3 we investigate the question how the firing time of such a neuron relates to the weighted sum of its temporally encoded inputs. The technical details of the simulations described in Section 9.2 and 9.3 are again given in Chapter 11.

We will conclude this chapter by showing that due to these positive results such biologically more realistic neurons can be used for pattern analysis on the basis of the Maass-construction (Section 9.4) and for realizing self-organizing behavior (Section 9.5) as described in Chapter 7.

Parts of Section 9.2 and 9.3 were presented in [Ruf, 1997], a summary of Section 9.4 in [Ruf, 1996].

9.1 The GENESIS Simulator

The ”general neural simulation system” (GENESIS) has been developed at the California Institute of Technology by Bower, Beeman et al. [Bower and Beeman, 1998]. It allows the simulation of biologically realistic neurons using compartmental modeling: the neuron is divided into a finite number of interconnected components, usually numerous ones for the dendritic tree, one for the soma and one for the axon. The axon might also be constructed from several compartments; however, usually it is sufficient to treat it as a simple delay line. In order to achieve a realistic model of the neuron, each compartment has to be small enough to have approximately a constant potential. For neurons with complex dendritic trees this requires a large number of compartments (e.g. a recently published GENESIS model of a Purkinje cell consists of 4550 compart-

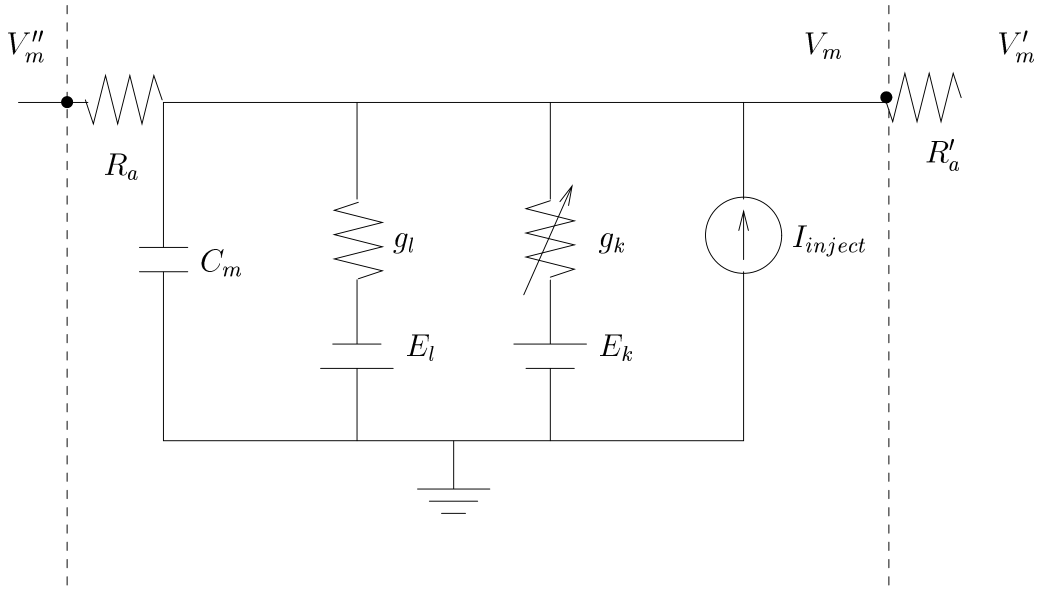

Figure 9.1: Equivalent circuit for a basic neural compartment (redrawn from [Bower and Beeman, 1998], p. 11).

ments [de Schutter and Bower, 1994]). For the simulation of networks of such neurons one usually has to model these neurons with a much smaller number of compartments due to the limitation of computation time. It has turned out that these reduced models are still able to capture many important aspects of experimentally observed behavior. For example, the GENESIS simulation of the olfactory cortex using 4500 neurons [Wilson and Bower, 1992] models various properties like the electroencephalogram (EEG). Each neuron consisted here only of at most 5 compartments.

As already mentioned in Section 2.2.2, a small piece of a membrane, having equivalent potential, can be described by an equivalent circuit, forming one compartment. The equivalent circuit for a basic neural compartment is shown in Figure 9.1. The voltage-independent ion-channels are included by the g_l-E_l branch, modeling the membrane leakage. Channels with variable conductances (like the Hodgkin-Huxley channels described in Section 2.2.1) are condensed here as one g_k-E_k branch. Moreover, the circuit includes the membrane capacitance C_m and allows the injection of an external current I_{inject}. The compartment is connected with two other compartments of potential V'_m and V''_m by axial resistances R'_a respectively R_a.

One can easily show (see e.g. [Bower and Beeman, 1998]) that the membrane potential V_m of this compartment is given by:

$$C_m \frac{dV_m}{dt} = \frac{E_m - V_m}{R_m} + \sum_k ((E_k - V_m) \cdot G_k) + \frac{V'_m - V_m}{R'_a} + \frac{V''_m - V_m}{R_a} + I_{inject} \qquad (9.1)$$

where the sum over all k represents a sum over the different types of ion channels within the compartment. Hence the computation of V_m requires solving the differential equation (9.1), which is coupled with the corresponding equations for V'_m and V''_m. In addition the current conductances g_k have to be computed; hence the simulation of a multi-compartmental model is computationally very expensive. GENESIS employs different numerical techniques for solving the differential equations, where they are replaced by difference equations, which are solved at discrete time intervals. With smaller time intervals a greater accuracy is possible, but at the cost of higher computational costs. Hence a proper choice of this so-called integration time step size is important.

A synaptic compartment is modeled by including a g_{syn}-E_{syn} branch in the circuit shown in Figure 9.1. The arrival of a presynaptic spike at the synapse causes the activation of chemically gated channels, having the reversal potential E_{syn}. The corresponding conductance g_{syn} is time-dependent, resulting in PSPs as shown in Figure 2.2. It is modeled by the generalized α-function, which is given by

$$g_{syn}(t) = \frac{A \cdot g_{max}}{\tau_1 - \tau_2}(e^{-t/\tau_1} - e^{-t/\tau_2}), \qquad \text{for } \tau_1 > \tau_2 \qquad (9.2)$$

where A is a normalization constant such that g_{max} is the maximal conductance of this synapse. For proper choices of τ_1 and τ_2 typical PSPs like the ones shown in Figures 2.2 and 8.12 can be modeled easily. A synapse produces an EPSP, if $E_{syn} > \Theta$ and if it increases the conductance. If $E_{syn} < \Theta$ and if it increases the conductance, it produces an IPSP.

9.2 Computing Linear Functions

A weighted sum respectively a linear function can be computed by the Maass-construction as described in Chapter 5. Two main assumptions are used for this construction: the neurons have to be of the leaky integrate-and-fire type and the initial segment of the PSPs has to be linear. The question arises whether biologically more realistic models of neurons, which only approximate these assumptions, can perform this computation. As in Chapter 5 we consider again a neuron v which receives input from neurons $u_0, \ldots, u_m$, where each u_i fires at time $t_i \in [T_{in} - \gamma, T_{in}]$ with $t_0 = T_{in}$. The resulting PSPs make v fire at a time t_v. With the help of GENESIS we want to find out if the time t_v still depends in a linear way on the inputs s_i where nonlinear effects could be caused by

- the interaction of different EPSPs in the dendritic tree

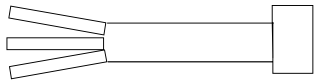

Figure 9.2: Compartmental model of a neuron as used in Section 9.2.

- the Hodgkin-Huxley kinetics at the soma of neuron v
- various types of noise.

Neuron v was built out of $m+3$ compartments: the soma, the root of the dendritic tree and for each of the $m+1$ inputs a separate dendritic branch (see Figure 9.2). Each input neuron was made fire exactly once at a random time point within the coding interval of size γ. All weights were set to random values from a certain interval. This initialization included w_0, hence we consider here the somewhat more general case where all weights may have arbitrary values.

All weights were then multiplied with a scaling factor κ, in order to guarantee that the incoming EPSPs suffice to make v fire. Several trials with one fixed weight setting but varying input firing times t_i were performed in order to find out if the resulting output firing times depend approximately in a linear fashion on these t_i. A precise linear behavior cannot be expected due to the above-mentioned non-linear effects. We want to know to what extent those effects cause basically only some Gaussian distributed background noise. We tried to characterize the dependency through a linear function determined via linear regression and measured the quality of the linear regression with the so-called coefficient of determination. It is defined as the ratio between the variance of the output values estimated by the regression and the variance of the observed output value (see, e.g. [Sachs, 1984], p. 389).

In our context, this means the following: assume that k experiments with one fixed weight setting had been performed such that v had fired for the ith input vector at time t_i. With the weights computed from the linear regression, the firing time $\tilde{t}_i$ had been predicted. Then the coefficient of determination β is given by

$$\beta = \frac{\sum_{i=1}^{k}(\tilde{t}_i - \bar{t})^2}{\sum_{i=1}^{k}(t_i - \bar{t})^2} \tag{9.3}$$

where $\bar{t}$ is the mean over all t_i. One can easily show that $0 \leq \beta \leq 1$. The closer β is to one, the closer the linear regression explains the observed values.

The quality of the results depends strongly on γ and the size of the weights. Figure 9.3 shows, how the coefficient of determination β depends on the weight scaling κ with $\gamma = 1.5\,ms$. For this experiment, neuron v received 10 synaptic inputs, for each synapse the maximum sodium conductivity (i.e. the weight) was set initially to a random value between 0 and $0.31\,nS$. 100 experiments with varying input firing times were performed and then the resulting regression function and its coefficient of determination computed. These experiments were repeated with different values of κ, i.e. all weights were multiplied with a scaling factor, which was varied between 1 and 100. v did not fire at all for very small values of κ. Hodgkin-Huxley kinetics at the soma generally deteriorate the linear behavior. It can be seen in Figure 9.3 that even in this case due to the small γ the behavior of v can be described as linear. The coefficients determined by the linear regression approximated the weights very well. E.g. for $\kappa = 20$ the variance of the ratios between the real weights and the predicted weights was less than $1.6 \cdot 10^{-4}$ (see Chapter 11 for details).

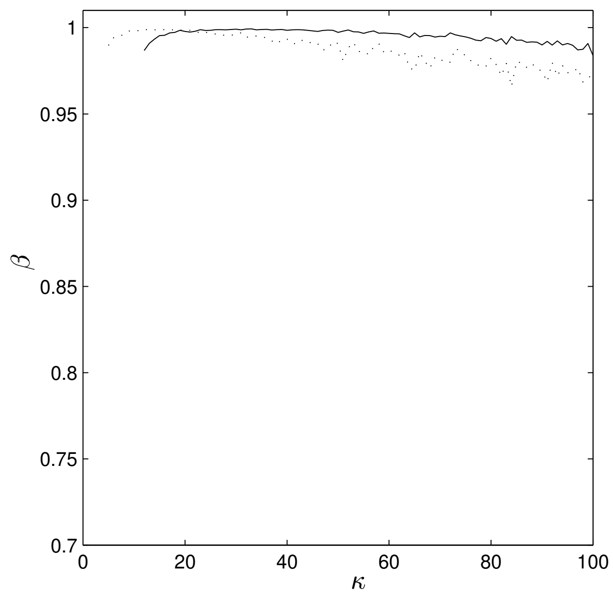

Figure 9.3: Coefficient of determination β in dependence of the weight scaling factor κ with $\gamma = 1.5\,ms$ for the experiment described in Section 9.2. The dotted (solid) line shows the results for the case where the soma contained (no) Hodgkin Huxley channels.

These results indicate that a linear computation is possible for a small coding interval. However, it is dubious if biological neurons can fire with such a high precision. If one relaxes the timing constraints on the input neurons to $\gamma = 3\,ms$ (approximately the length of the rising phase of the EPSPs in these experiments) the range of weights where the computation of the neuron can be described as linear becomes smaller. This is due to the facts that the firing time of v is more strongly influenced by the non-linear segments of the EPSPs and that for strong weights the peak of the resulting EPSP depends sublinearly on the weights. For small factors which suffice to make v fire, no linear dependency could be observed, because the potential of v increases in this case rather slowly. Thus many EPSPs will have reached their non-linear saturation segment when v fires. Figure 9.4 shows that nevertheless linear computations are possible within a wide range of weight values. It also indicates that not only for large, but also for very small weight values, β decays considerably. This may be explained as follows: if one considers the rising segment of an EPSP as an approximation of a linear function, then the initial and final parts of the rising segment cause the largest errors of the approximation. Thus, if all weights are very small, the resulting potential at the soma of v rises not very steeply when reaching its threshold such that in this case the initial and final rising segment of the EPSPs have a stronger influence on the firing time. Hence they cause for small κ non-linear effects.

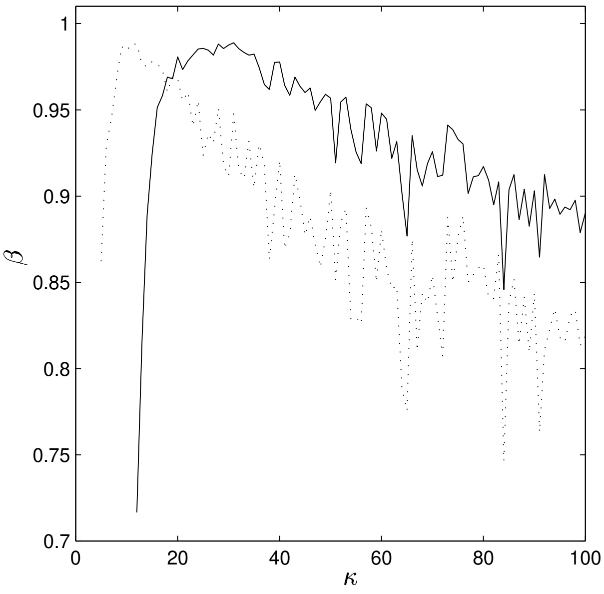

Figure 9.4: As in Figure 9.3 but with $\gamma = 3\,ms$.

A γ which is larger than the length of the rising segment of the EPSP can also be achieved by assuming for example that a presynaptic neuron forms several synapses to the postsynaptic neuron with different delays and weights, such that if the presynaptic neuron fires the changes of the postsynaptic potential result in a long linear increasing segment (see [Maass and Natschläger, 1997] for more details).

For the construction presented in this section a proper weight scaling is needed in order to guarantee that v actually fires. However, similar as in Chapter 6, this assumption is not really necessary if one makes use of sufficiently many auxiliary EPSPs which increase the resting potential of v by a constant amount, such that a smaller input to v

now suffices to exceed its threshold.

The results in this section indicate that a neuron can compute in a very simple and straightforward way a linear function in temporal coding. In the next section we investigate how the weighted sum of the temporally encoded inputs is related to the firing time of the output neuron and in how far this can be used to approximate arbitrary functions.

9.3 Activation Functions

As described in Chapter 5, the Maass-construction allows the simulation of a sigmoidal gate with SNNs. It requires that the neuron v, which computes this function, receives input from some auxiliary neurons. In this section we investigate the question, whether such a neuron, which receives one input spike from each input neuron u_i within a certain coding interval in the same fashion as in the last section, i.e. which does not receive any reference spike or any auxiliary inputs, has an activation function, i.e. whether there is a functional dependence between the weighted sum $\sum w_i s_i$ and the firing time of v and of what type it might be.

First we observe that if all but one inputs fire at fixed times, the activation function, which results from varying the one "free" input, is basically determined by the shape of the EPSP: given a neuron v with n inputs, fixed weights $w_i, 1 \leq i \leq n$ and fixed inputs $s_i, 2 \leq i \leq n$ only s_1 may be varied. Figure 9.5 shows the result of a GENESIS simulation with $n = 10$. Similar as in Section9.2 each input was again on a separate branch of the dendritic tree and s_1 was allowed to vary within a coding interval of size $\gamma = 9\,ms$. 200 trials were executed, where the output value was computed each time by subtracting the current firing time t_v of v from the largest firing time t_v^{max} occurring in this experiment. Note that the output values shown in Figure 9.5 equal zero for small weighted sums, since in this case s_1 is very small, which means that the corresponding input neurons fires very late, such that its EPSP does not influence the firing time of v.

When all the weights are fixed and all s_i may vary within γ, where γ is of approximately the length of the rising segment of the EPSPs, the firing time of v should depend on $\sum w_i s_i$ approximately in a linear fashion. As mentioned in Section 9.2 this can be only true, if the weights are not too large or too small (both cases would cause nonlinear effects). An example is shown in Figure 9.6.

If one varies both the weights and the inputs, the choices of the intervals where the inputs respectively the weights are chosen from are critical. For γ being approximately of the length of the rising phase of the EPSP and weights chosen from an interval being within the range where the slope of the EPSP is influenced linearly by the corresponding weight, one can no longer say without any further assumptions that neuron v computes some function in the sense described above: It is possible that for two different weight- and input-vectors $\mathbf{w}, \mathbf{s}$ and $\mathbf{s}', \mathbf{w}'$ with $\sum w_i s_i = \sum w_i' s_i'$ the resulting firing

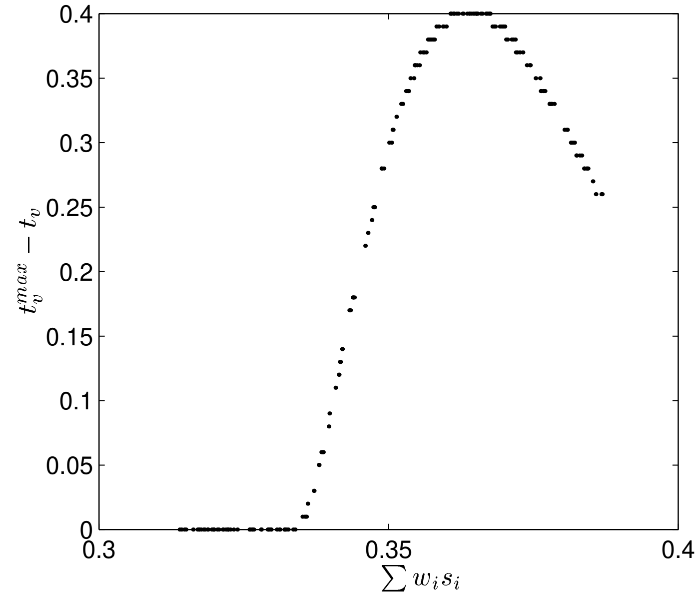

Figure 9.5: Dependence of the output value of neuron v from the weighted sum $\sum w_i s_i$, where only one input neuron varied its firing time.

times of v differ considerably. However, if one increases the lower bound of the weight interval (we have seen in Section 9.2 that small weights cause non-linear effects) and uses a γ being smaller than the rising phase of the EPSPs, one gets an approximately linear dependence, similar as in Figure 9.6.

All experiments in this section were performed without Hodgkin-Huxley channels. As in Section 9.2 including these channels does simply reduce the range of possible parameters and adds some noise to the results, but the type of computations described above is still possible.

9.4 Pattern Analysis

Various types of neural networks are known to be suitable for pattern analysis tasks, most of them working on the basis of firing rates. Especially for fast information processing a solution based on temporal coding is desirable. Here we describe how the Maass-construction can be used for pattern analysis with biological realistic neurons and investigate how reliable these neurons perform on this approach.

Similar as in Chapters 7 and 8 we consider a network of two layers, where each neuron of the first layer is connected to each neuron of the second layer. It has m input neurons in the first layer and n output neurons in the second layer. We assume again

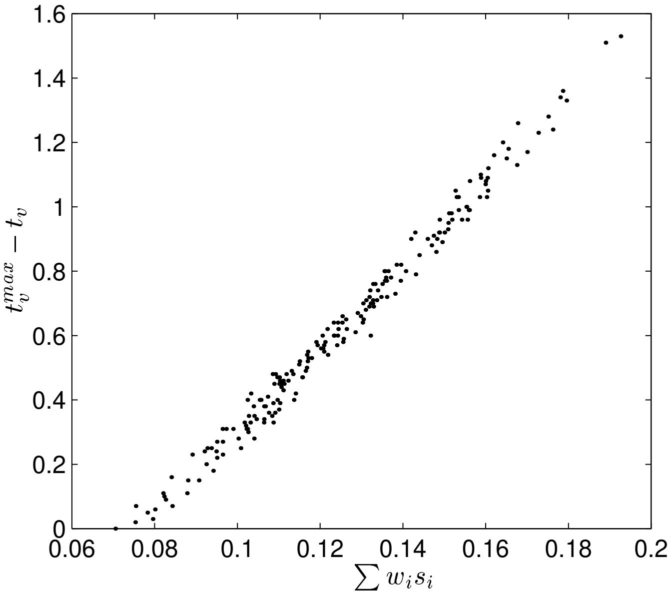

Figure 9.6: As in Figure 9.5, but where all input neurons could fire at random times within a coding interval of length $\gamma = 3\,ms$.

that the assumptions of the Maass-construction are fulfilled.

The system has stored n patterns $\mathbf{s}^1, \ldots, \mathbf{s}^n \in [0, K]^m$ for some $K > 0$, each consisting of m components. We assume that these patterns are pairwise orthogonal, i.e.

$$\mathbf{s}^i \cdot \mathbf{s}^j = \sum_{k=1}^{m} s_k^i s_k^j = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases} \qquad 1 \leq i, j \leq n \tag{9.4}$$

The patterns are stored in the weights of the synaptic connections between the two layers such that the weight w_{ij} for the connection from the ith input neuron to the jth output neuron represents the ith component of the jth stored pattern $\mathbf{s}^j$, i.e. $w_{ij} = s_i^j$. The system receives as input linear combinations of the stored patterns

$$\tilde{\mathbf{s}} = \sum_{k=1}^{n} \mu_k \mathbf{s}^k \tag{9.5}$$

where the $\mu_k \in [0, \gamma], 1 \leq k \leq n$ have to be chosen such that $\tilde{\mathbf{s}} \in [0, \gamma]^m$. This input is presented to the input neurons in temporal coding in the same way as for the Maass-construction.

The jth output neuron v_j is made fire by the input $\tilde{\mathbf{s}}$ at a time t_j which is (similar as in Equation (5.7)) given by:

$$\begin{aligned} t_j &= T_{out} - \frac{1}{\lambda}\sum_{i=1}^{m} w_{ij}\tilde{s}_i \\ &= T_{out} - \frac{1}{\lambda}\sum_{i=1}^{m} s_i^j \sum_{k=1}^{n} \mu_k s_i^k \\ &= T_{out} - \frac{1}{\lambda}\sum_{k=1}^{n} \mu_k \mathbf{s}^j \cdot \mathbf{s}^k \\ &= T_{out} - \frac{1}{\lambda}\mu_j \end{aligned}$$

due to the orthogonality assumption (9.4). As in Section 5.1 we have again $\lambda = \sum_{i=0}^{m} w_{ij}$, $1 \leq j \leq m$. Hence the firing time of the jth output neuron represents the strength of the jth pattern in $\tilde{\mathbf{s}}$.

In Section 9.2 we have seen that a computation of a weighted sum is possible even with biologically more realistic neurons than leaky integrate-and-fire neurons. These results motivate the question in how far the type of pattern analysis described above is possible with such models. Here it is essential that the firing times of various output neurons are comparable. We used again GENESIS, where the output neurons were constructed as described in Section 9.2.

We have restricted the system to positive values. However, by using IPSPs, it is also possible to store patterns with negative components. In this case one has to find a way to temporally encode negative input values for the input vector $\tilde{\mathbf{s}}$. This can be achieved, for example, by considering a smaller γ, where the reference spike occurs in the middle of the linear segment of the PSP, such that spikes occurring after the reference spike are considered as a negative input.

The scaling of the weights is a crucial point in our model as already explained in Section 9.2. We were able to show that the system behaves within quite a wide range very stable and reliable despite of those constraints especially for the case that all stored patterns consist only of positive reals. When using also IPSPs there are stronger non-linear effects in the integration of the PSPs at the soma.

Figure 9.7 shows the result of a GENESIS simulation with $m = 6$ and $n = 4$. The coefficients of the presented linear combinations are determined by the output neurons with a high precision, although for this simulation even Hodgkin-Huxley channels were present at the soma. The errors were less than $1\,ms$.

This model has several essential advantages: The whole computation can be performed by biological neurons in a very short time (approximately $10\,ms$). Since the output is encoded in the same fashion as the input one could imagine a sequence of such systems, which first recognize low-order patterns and later high-order patterns. Hence complex analysis tasks can be solved by systems in a time range which is known to suffice also for various neural biological systems.

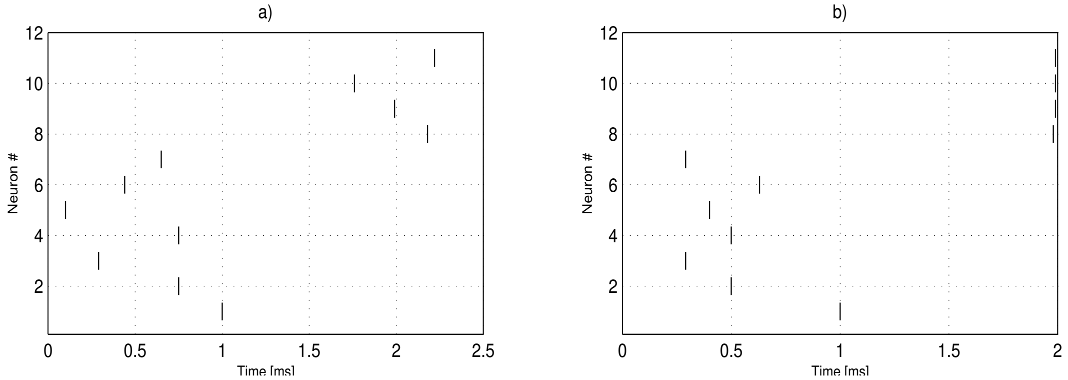

Figure 9.7: Pattern analysis with an SNN of $m = 6$ inputs (neurons #2 to #7 plus neuron #1 for producing the reference spike at $t_0 = 1\,ms$) and $m = 4$ outputs (neurons #8 to #11). The input was a linear combination of the stored patterns. For Figure a), we chose $\mu_1 = \mu_4 = 0.5$, $\mu_2 = 2 \cdot \mu_1$ and $\mu_3 = 3 \cdot \mu_1$, for Figure b) $\mu_i = 1.0$, $1 \leq i \leq 4$.

For certain pattern analysis tasks the coefficients μ_j may be time-dependent. Such a dependency can be handled easily within this network, because a repeated presentation of input patterns $\tilde{\mathbf{s}}$ with varying μ_j would cause corresponding shifts in the firing times of the output neurons. These shifts may then be processed in subsequent layers.

It is not necessary that the orthogonality-condition for the stored patterns is precisely fulfilled. Pattern analysis tasks take frequently the similarity between the pattern into account: certain patterns may be very similar to each other and very dissimilar to other patterns. This may be reflected in the inner product between those vectors. If we assume that $\mathbf{s}^i \cdot \mathbf{s}^k = 0$ only for two very dissimilar patterns $\mathbf{s}^i$ and $\mathbf{s}^k$ but allow that $\mathbf{s}^i \cdot \mathbf{s}^j > 0$ for two related patterns $\mathbf{s}^i$ and $\mathbf{s}^j$ then the presence of $\mathbf{s}^i$ enhances the output $\mathbf{s}^i$ and vice versa, which appears to be also the case in certain biological systems.

One example, where this kind of pattern analysis is relevant to biological neural systems is the analysis of odors in the olfactory system. One important aspect of many sensory systems like the visual and auditory system is the construction of a map of the spatial location of a stimulus. In contrast to this, odors are not located due to the stimulus location on the layer of the sensory neurons. It has turned out that instead the spatial activity patterns are used for the analysis of mixtures of odors. The sensory neurons project to the olfactory bulb, where a stimulation with one odor activates certain regions of the olfactory bulb. Different odors activate different sensory neurons thus activating different regions in the olfactory bulb (which may, however, overlap). Sensory neurons may be highly specialized to certain odors, projecting to specialized regions of the bulb or more generalized sensory neurons which project to several regions of the bulb. Hence a mixture of odors activates several regions in the olfactory bulb (for details see e.g. [Shepherd, 1994]).

9.5 Self-organizing Maps

Chapter 7 showed how SNNs can simulate a SOM in temporal coding. It is not obvious how the nonlinearities described in Section 2.2 influence the learning quality where several thousand iterations of one learning rule are made. By executing the SOM simulations for one- and two-dimensional input patterns described in Section 7.3 with biologically more realistic neurons as described above through GENESIS it has turned out that such networks are still able to produce the characteristic self-organizing behavior [Fakkles, 1998].

10 Conclusions and Outlook

We have seen that SNNs are computationally very powerful and fast neural networks, whose consideration and analysis are important both from the biological as well as from the computational point of view. In this thesis we have focussed on temporal coding where no time-consuming sampling of a firing rate is necessary. We have seen that Hebbian learning, frequently used for ANNs, may also serve as a basis for realizing learning with SNNs using temporal coding. Moreover, it has turned out that various types of ANNs which are known to be very powerful can be adopted for SNNs in the temporal domain. In the case of radial basis functions we were able to get additional information by considering firing times of neurons, which gave rise to additional features as clustering. For our constructions we mainly focussed on the LIFN model, but the results in Chapter 9 indicate that due to the noise robustness of the constructions also other biologically more realistic models might be used.

The constructions presented in this work also seem to be suitable for silicon implementations in a closely related form for pulse-coded VLSI [Murray and Tarassenko, 1994], where one builds hardware for analog computation encoding information in pulses. It allows pulse phase modulation where a value is represented by the temporal delay between binary pulses with fixed amplitude, which closely resembles the way of temporal coding using action potentials of spiking neurons which we studied here. There exist several approaches for VLSI implementations of Hebbian learning with spiking neurons as described in Section 2.4 [Häfliger et al., 1997; Roth et al., 1997a,b]. Our methods are for example suitable for an implementation with the NESPINN, the neurocomputer for the simulation of spiking neurons [Roth et al., 1997b] allowing the fast and parallel realization of self-organization and clustering, to name just two possible applications.

It has been shown in this work how various types of ANNs can be implemented with SNNs using temporal coding. There are several other types of ANNs which are suitable for an adaptation for SNNs as well: in [Maass and Natschläger, 1997] it has been shown how SNNs can emulate arbitrary Hopfield nets in temporal coding, which is also based on the Maass-construction. For a long time the error-backpropagation algorithm was considered as biologically very unrealistic since it requires within a multi-layer feedforward network the backpropagation of an error signal from the output layer back to the input layer. However, recently it has been shown in [Fitzsimonds et al., 1997] that in hippocampal neurons the induction of long-term depression at some synapse causes the back propagation of depression to input synapses on the dendrite of the *presynaptic* neuron. This means that information also flows in addition in the opposite axon-terminal-to-dendritic direction. With the use of such an error-signal the realization of backpropagation through SNNs seems to be possible (see also [Nicoll and Malenka, 1997]).

For the results given in Chapters 5 – 8 the simplifying LIFN-model was used. As shown in Chapter 9, many results still hold for biologically more realistic models. Future work will show how other features of biological neurons than the ones discussed in Chapter 9 influence the quality of our methods and the computational power of such neurons in general. While for example noise, as it may occur in the response- and threshold functions obviously causes a deterioration, other features like the linearity results for dendritic trees given in [Cash and Yuste, 1998] may even cause an improvement.

This work is only one step within the exploration of possibilities how to compute and learn with SNNs. Since SNNs are computationally more powerful than ANNs it appears to be very promising to design new algorithms especially for SNNs which make efficient use of the additional possibilities like the timing of spikes. Moreover, it seems very likely that the integration of additional features of biological neurons increases the computational power of SNNs. For example, synapses are known to be highly probabilistic, i.e. the arrival of a spike at the presynaptic terminal causes only with a certain probability (which may be history dependent) a postsynaptic spike (see for example [Katz, 1966; Dobrunz and Stevens, 1997]). One possibility for using these features for analyzing spike trains with such synapses has been presented in [Maass and Zador, 1998]. Furthermore, the computational power of SNNs may be enhanced by making use of other synaptic features like synaptic depression and facilitation [Abbott et al., 1997] or the additional communication possibilities provided through neuromodulators, which may also act on a global level.(see, e.g., [Shepherd, 1994]).

Our methods rely mainly on single spikes, i.e. each input neuron is allowed to fire only once within the coding interval. However, the presentation of an input vector may just happen within a small time window of a longer lasting computation. For example, one may consider the case where all neurons of the SNN fire with the same frequency, but phase-shifted, as suggested in [Hopfield, 1995]. Then, in the case of the Maass-construction, a weighted sum is computed within each period. A phase-shift for some input neurons yields a phase-shift of the output neuron.

11 Technical Appendix: Description of the Simulations

In this chapter we give more details about our simulations described in Chapters 7 – 9. Besides the concrete choices of parameters for our simulations we also summarize some more implementation details which have not been mentioned in the previous chapters.

11.1 SOM Experiments in Chapter 7

Given a training set S and a layer of n competitive neurons with neighborhood matrix $(m_{kj})_{1 \leq k,j \leq n}$, we first initialize the weights as follows: the weight vector $\mathbf{w}_j$ for the feedforward weights of neuron v_j, $1 \leq j \leq n$ is set to the mean pattern $\overline{\mathbf{s}}$ of S. In contrast to the initialization for the standard formulation of the SOM, one has to add some noise to $\mathbf{w}_j$, because otherwise all competitive neuron would fire at the same time and no learning would take place. We set w_{ij} to a random value from $[\overline{s}_i - 0.1, \overline{s}_i + 0.1]$ for every i, j. Then the resulting weight vector is normalized and the weight for the additional neuron u_0 computed such that $\sum_{i=0}^{m} w_{ij} = \lambda$ for $1 \leq j \leq n$. The constant λ was set sufficiently large in order to guarantee that every competitive neuron actually fires. For our simulations we made the quite arbitrary choice $\lambda = 3.0$. The lateral weights were computed from the neighborhood matrix by a simple linear transformation in order to adopt the positive integer values to positive and negative values within some bounded interval. We chose $\tilde{w}_{kj} = lw \cdot (-1/m_{max} \cdot m_{kj} + 0.25)$, $1 \leq k, j \leq n$, where m_{max} is the largest element from the neighborhood matrix.

In the main program one $\mathbf{s}^l \in S$ is selected for every learning cycle and the input neurons are made fire at the corresponding times. Assuming that all firings of the competitive neurons take place while the incoming PSPs are in their linear segment (otherwise one could easily scale the input pattern) one can directly compute the firing time of the winner neuron. Then the firing times of the other competitive neurons are computed iteratively, also taking into account the input through the lateral connections of those competitive neurons which fired already. Since it is possible that some competitive neurons do not fire at all, this computation is performed only up to the time T_{out}, where the parameter T_{out} denotes the last (allowed) firing time.

With the knowledge of all firing times the weights are updated according to our learning rule (7.2). Then for every v_j the weight vector $\mathbf{w}_j$ (excluding w_{0j}) is normalized and w_{0j} adopted to get again $\sum_{i=0}^{m} w_{ij} = \lambda$. Finally the learning rate η is slightly decreased (we chose $\eta^{new} = 0.995 \cdot \eta^{old}$) and the lateral weights are reduced by a constant value ld.

In the first simulation described in Section 7.3 for learning a one-dimensional pattern, we used the following parameters: there were 10 competitive neurons and the training-set S was of the same size. The patterns were uniformly distributed over $[0.1, 0.9]$ where we assumed that every competitive neuron fires while its incoming PSPs are still in their linear segment (otherwise the coding interval would have to be scaled with a factor smaller 1.0). The neighborhood matrix was chosen such that $m_{kj} = |k - j|$, $1 \leq k, j \leq n$. Furthermore, we set $\eta = 0.1$, $\Theta = 1.5$, $ld = 0.0005$ and $lw = 2$. Then 4000 learning cycles were executed, producing the results shown in Figure 7.2.

In the second experiment in Section 7.3 a grid of 5×5 competitive neurons was trained on a set of 1000 input pattern uniformly distributed over $[0, 0.7]^2$. The neighborhood matrix was computed using the Manhattan distance for a 5×5 grid. The other parameters were set to $\eta = 0.1$, $\Theta = 1.5$, $ld = 0.0005$ and $lw = 3$. The results shown in Figure 7.4 were obtained after 8000 learning cycles.

11.2 RBF Experiments in Chapter 8

All RBF neurons were of the leaky integrate-and-fire type, where the following parameters were used throughout all experiments described in Chapter 8: the coding interval was set to $10\,ms$, the length of one learning cycle to $50\,ms$. During learning the weights were modified with a learning rate $\eta = 0.04 \cdot w_{max}$. The set of available delays throughout our experiments was $D = \{1\,\mathrm{ms}, \ldots, 16\,\mathrm{ms}\}$.

The EPSPs were computed by the α-function (9.2) with $\tau_1 = \tau_2 = 3\,ms$ except for Section 8.4, where we employed variable EPSPs. τ_1 and τ_2 were chosen in this case from $\{0.5\,ms, 1.0\,ms, \ldots 8\,ms\}$. The threshold of each RBF neuron was set to $\Theta = 1.0$ for all simulations and the absolute refractory period to $30\,ms$ such that each RBF neuron fires at most once within a learning cycle. The lateral (inhibitory) synapses had a weight of -4.0 and no delay. Again we set $\tau_1 = \tau_2 = 3\,ms$. For the slow self-inhibition, we chose a weight of -0.2 and a delay of $1\,ms$. Since the resulting PSPs should have a much longer inhibitory impact on the neuron we set $\tau_1 = \tau_2 = 60\,ms$ such that the inhibition lasts a bit longer than the next learning cycle.

All feedforward weights are excitatory and chosen from an interval $[w_{min}, w_{max}]$ with $w_{min} = 0.0$ and $w_{max} = 0.026$ for all simulations except in Section 8.3, where we chose $w_{max} = 0.038$. The estimates for w_{max} were obtained by observing that after an RBF neuron has learned a certain center, there are for each input only a few neighboring active delays, i.e. which have a weight close to w_{max}. The number of these active delays depends on the shape of the learning function $L(\Delta t)$ and is for our choice of $L(\Delta t)$ (see below) approximately determined by $q = \lfloor p/(d^l - d^{l-1}) \rfloor$ where p is the length of the segment of $L(\Delta t)$ for which $L(\Delta t) > 0$ and $d^l - d^{l-1} = 1\,ms$. The peak of the membrane potential of an RBF neuron is largest for an input vector corresponding to the center of this RBF neuron and smaller for other input vectors. w_{max} should be large enough such that the RBF neuron fires for all input vectors, hence we chose for $w_{max} = k \cdot \Theta/(m \cdot q)$ with $k > 1$. We set $k = 1.02$ except for the case when the cluster was formed in some subspace (Section 8.3), where we set $k = 1.20$. The choice of k is not very important when the learning procedure is stopped after the RBF neurons have converged. Only if learning is continued afterwards for a long time, the system may became instable due to a w_{max} being too large, as described in Section 8.5, which makes a good choice for k necessary.

The feedforward weights were now initialized as follows: they were set to random values from $[w_{min}, w_{max}]$, where we made sure that they had a sufficient variance (such that different RBF neurons behave different already at the beginning of the learning procedures) and such that they actually fire for each input vector.

The learning function was chosen as

$$L(\Delta t) = \begin{cases} (1-b) \cdot \exp\left(\frac{-(\Delta t - c)^2}{\beta^2}\right) + b, & \Delta t \in [-\tilde{\Delta}, \tilde{\Delta}] \\ 0, & \text{otherwise} \end{cases}$$

with $\tilde{\Delta} = 15\,ms$, $b = -0.11$, $\beta = 1.11\,ms$ and $c = -2.0\,ms$. These values were used for all simulations except for Section 8.3, where a slightly modified learning function with $c = -1.5\,ms$, $\beta = 1.67\,ms$ and $b = -0.2$ was taken.

The centers of the clusters were initialized as follows: if there were p clusters, we partitioned the coding interval into p intervals of equal length. Then for each coordinate of the input space each of these intervals was assigned to exactly one cluster and the corresponding coordinate of the center of this cluster set to the middle of this interval. In addition some Gaussian noise with standard deviation 0.2 was added to the coordinates. Then the clusters themselves were generated by assuming a Gaussian distribution around the center with a standard deviation of 0.1.

For the simulations in Section 8.3 we used a piecewise linear saturation function $S(w)$ as shown in Figure 11.1.

11.3 Linear Functions Experiments in Chapter 9

All simulations were performed with GENESIS, where most parameters (e.g. specifications of the ionic channels, geometry of compartments, etc.) were chosen as in the standard GENESIS tutorials (see Section 11.4). As mentioned in Section 9.2, the soma compartment was linked to a compartment representing the root of the dendritic tree. This was connected to n compartments, which were 20% shorter and had a diameter being 25% smaller than the root compartment. Each of these compartments received

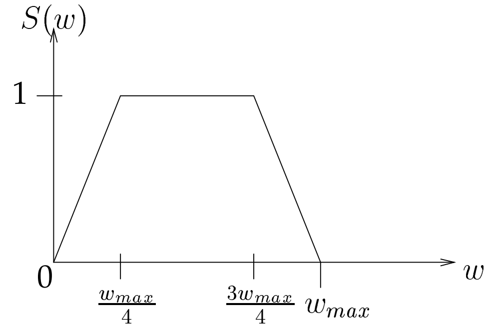

Figure 11.1: Saturation function used for the simulations in Section 8.3.

through an excitatory synapse spikes from an input neuron. The initial maximal conductance for sodium ions was set to $0.31\,nS$. The weight of this synapse is the factor, with which this conductance is multiplied. The simulation time step was set to $0.01\,ms$.

For the simulations in Section 9.2 the weights were set to random values from $[0, 1]$; for the plots they were set to $\mathbf{w}$ =(0.911, 0.543, 0.732, 0.054, 0.658, 0.275, 0.572, 0.125, 0.351, 0.218). The weights were then scaled with a factor κ, which was initially set to 1.0 and then increased by one until it reached 100.0. For each κ, all inputs were made fire at random times in the interval $[0, \gamma]$ and the resulting firing time recorded (for small κ, the neuron did not fire at all). For each κ linear regression with the measured firing times was performed and the coefficient of determination computed. Furthermore, the resulting weight predictions $\tilde{\mathbf{w}}$ were compared with the real weight-vectors $\mathbf{w}$ of the simulations. For $\kappa = 20$ this resulted in the vector $(\tilde{w}_1/w_1, \ldots, \tilde{w}_{10}/w_{10})$ =(0.2038, 0.2269, 0.2147, 0.1931, 0.2180, 0.2279, 0.2220, 0.2352, 0.2277, 0.2238) having a variance of $1.5947 \cdot 10^{-4}$. The step-rate of the internal clock for the simulations was set to $10^{-5}sec$.

The neurons in Section 9.3 were constructed using the same parameters. For the plots of Figure 9.6 and 9.5, weights were chosen from the interval $[3.2\,nS, 4.6\,nS]$.

The examples for realizing pattern analysis with SNNs were achieved with the following data: the weights were set according to the weight matrix

$$\mathbf{w} = \begin{pmatrix} 3.333 & 5.0 & 0.0 & 5.0 & 0.0 & 0.0 & 0.0 \\ 6.262 & 0.0 & 7.071 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.590 & 0.0 & 7.071 & 0.0 & 6.0 & 3.742 & 0.0 \\ 6.262 & 0.0 & 7.071 & 0.0 & 0.0 & 0.0 & 7.071 \end{pmatrix}$$

where the jth row $\mathbf{w}_j$ gives the weight vector for the jth output neuron. The weights for u_0 were chosen such that they fulfill the orthogonality condition and the conditions of the Maass-construction: for each $\mathbf{w}_j$ we have $\mathbf{w}_j = \sum_{i=0}^{m} w_{ij} = 13.333$ and $\sum_{i=1}^{m} w_{ij}^2 = 50.0$ For the first experiment (Figure 9.7a) the input vector was $\tilde{\mathbf{s}} =$ (2.5, 7.071, 2.5, 9.0, 5.613, 3.535) $\cdot 10^{-4}\,ms$ according to a vector $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3, \mu_4) =$ (0.5, 1.0, 1.5, 0.5) $\cdot 10^{-4}$. Neuron u_0 fired at time $10^{-3}\,ms$.

For the second experiment (Figure 9.7b) we chose $\boldsymbol{\mu} = (1.0, 1.0, 1.0, 1.0) \cdot 10^{-4}$, resulting in $\tilde{\mathbf{s}} = (\,5.0, 7.071, 5.0, 6.0, 3.742, 7.071\,) \cdot 10^{-4}\,ms$. All weights were scaled by a factor of 10 for both experiments in order to guarantee that the output neurons fire.

11.4 Constants for the GENESIS simulations

The listing below gives a complete description of the constants chosen for the GENESIS simulation in Chapter 9. For a detailed description of the meaning of these variables, we refer to [Bower and Beeman, 1998].

```
// ===================================================================
//                         PHYSIOLOGICAL PARAMETERS
// ===================================================================
// All units in SI  and usually dimensions are given as well

//                       IONIC EQUILIBRIUM POTENTIALS
float   ENA             =       55.0e-3                      // V
float   ECL             =      -65.0e-3                      // V
float   EK              =      -90.0e-3                      // V
float   EREST           =      -70.0e-3                      // V
float   I_EREST         =      -70.0e-3                      // V

//                       ACTIVE COMPARTMENT POTENTIALS
float   EREST_ACT       =      -70.0e-3                      // V
float   ENA_ACT         =      115.0e-3 + EREST_ACT          // V
float   EK_ACT          =      -12.0e-3 + EREST_ACT          // V
float   ELEAK_ACT       =       10.6e-3 + EREST_ACT          // V


// squid giant axon - Conti et al.(1975),J.Physiol(Lond),248,45-82
// gNa = 4pS
// squid giant axon - Conti and Neher(1980),Nature(Lond),285,140-143
// gK = 18pS
// mouse spinal neuron - Mathers (1982),Int.Rev.Neurobiol,23,1-34
// gCl = 18pS
//                            UNIT CONDUCTANCES
//                            (Siemens/channel)
float   UNIT_GNA        =      12.0e-12        // S
float   UNIT_GCL        =      18.0e-12        // S
float   UNIT_GK         =       4.0e-12        // S
float   SUNIT_GNA       =       8.0e-12        // S
float   SUNIT_GK        =       4.0e-12        // S

// squid giant axon - Conti et al.(1975),J.Physiol(Lond),248,45-82
```

```
// rhoNa = 330 channels/um^2
// squid giant axon - Conti and Neher(1980),Nature(Lond),285,140-143
// rhoK = 72 channels/um^2
//                           CHANNEL DENSITIES
//                              (channels/m^2)
float   RHO_NA          =        60.0e12
float   RHO_CL          =        30.0e12
float   RHO_K           =        30.0e12
float   SRHO_NA         =       330.0e12
float   SRHO_K          =        72.0e12
float   IRHO_NA         =        10.0e12

//                           SYNAPTIC AREA
// Haberly and Presto(1986),J.Comp.Neurol,248,464-474
// basal spine d = .40um
// distal spine d = .74um
// presynaptic to dendritic spines d = .61
// presynaptic to dendritic shafts d = .89
// presynaptic to initial segment d = .89
//
// using synaptic contact area A = pi*d^2/4
//                              (m^2/synapse)
float   ASYN_LOCAL_NA   =       0.12e-12
float   ASYN_DISTAL_NA  =       0.43e-12
float   ASYN_CL         =       0.62e-12
float   ASYN_K          =       0.43e-12

float   IASYN_NA        =       0.29e-12

//                      ACTIVE AREA
// fraction of somatic area containing active channels
float   fAC             =       0.1


//                      PEAK CONDUCTANCE
//                      (Siemens/synapse)
float   I_GMAX_NA       =       UNIT_GNA * IRHO_NA * IASYN_NA
float   LOCAL_GMAX_NA   =       UNIT_GNA * RHO_NA  * ASYN_LOCAL_NA
float   DISTAL_GMAX_NA  =       UNIT_GNA * RHO_NA  * ASYN_DISTAL_NA
float   GMAX_K          =       UNIT_GK  * RHO_K   * ASYN_K
float   GMAX_CL         =       UNIT_GCL * RHO_CL  * ASYN_CL

float   SGMAX_NA        =       SUNIT_GNA * SRHO_NA       // S/m^2
float   SGMAX_K         =       SUNIT_GK  * SRHO_K        // S/m^2

//                      MEMBRANE PARAMETERS
float   RM              =       0.2             // ohm-m^2
```

```
float   RA              =       0.5             // ohm-m
float   CM              =       0.01            // F/m^2

//                      TYPICAL CELL DIMENSIONS

float   SOMA_D  =       20.0e-6                         // m
float   SOMA_L  =       20.0e-6                         // m
float   SOMA_A  =       PI * SOMA_D * SOMA_L            // m^2
float   SOMA_XA =       PI * SOMA_D * SOMA_D / 4        // m^2
//Root of Dendritic Tree:
float   DENDR_D =       2e-6                            // m
float   DENDR_L =       100.0e-6                        // m
float   DENDR_A =       PI * DENDR_D * DENDR_L          // m^2
float   DENDR_XA=       PI * DENDR_D * DENDR_D / 4      // m^2
//Branches of Dendritic Tree:
float   DENDB_D =       1.5e-6                          // m
float   DENDB_L =       80.0e-6                         // m
float   DENDB_A =       PI * DENDB_D * DENDB_L          // m^2
float   DENDB_XA=       PI * DENDB_D * DENDB_D / 4      // m^2

// PARAMETERS FOR SPIKE-GENERATOR

float   THRESHOLD=      -40e-3                           // V
float   ABS_REF=        3.0                             // s

/*
** VARIABLES USED BY ACTIVE COMPONENTS
*/
int     EXPONENTIAL     =       1
int     SIGMOID         =       2
int     LINOID          =       3
```

Bibliography

Abbott, L. F., Varela, J. A., Sen, K., and Nelson, S. B. (1997). Synaptic depression and cortical gain control. *Science*, 275:220–224.

Abeles, M. (1991). *Corticonics: Neural Circuits of the Cerebral Cortex*. Cambridge University Press.

Abeles, M., Bergman, H., Margalit, E., and Vaadia, E. (1993). Spationtemporal firing patterns in the frontal cortex of behaving monkeys. *Journal of Neurophysiology*, 70(4):1629–1638.

Arbib, M. (1989). *The Metaphorical Brain 2*. John Wiley and Sons, New York.

Blasel, G. and Obermayer, K. (1994). Putative strategies of scene segmentation in monkey visual cortex. *Neural Network*, 7:865–881.

Blum, E. K. (1972). *Numerical Analysis and Computation: Theory and Practice*. Addison-Wesley, Reading, Mass.

Bower, J. M. and Beeman, D. (1998). *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System (2nd edition)*. Springer-Verlag, Inc. Published by TELOS, New York.

Brown, T. H., Chapman, P. F., Kairiss, E. W., and Keenan, C. L. (1988). Long-term synaptic potentiation. *Science*, 242:724–728.

Brown, T. H. and Chattarji, S. (1995). Hebbian synaptic plasticity. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 454–459. MIT Press.

Cash, S. and Yuste, R. (1998). Input summation by cultured pyramidal neurons is linear and position-independent. *Journal of Neuroscience*, 18(1):10–15.

Choe, Y. and Miikkulainen, R. (1996). Self-organization and segmentation with laterally connected spiking neurons. Technical Report AI TR 96-251, Department of Computer Science, University of Texas at Austin.

Churchland, P. and Sejnowski, T. (1992). *The Computational Brain*. MIT Press, Cambridge.

de Schutter, E. and Bower, J. (1994). Responses of cerebellar Purkinje cells are independent of the dendritit location of granule cell synaptic inputs. *Proc. Natl. Acad. Sci*, 91:4736–4740.

Dobrunz, L. and Stevens, C. (1997). Heterogenous release probabilities in hippocampal neurons. *Neuron*, 18:995–1008.

Fakkles, E. (1998). Self-organizing maps with biologically realistic spiking neurons. Master's thesis, TU Graz.

Fitzsimonds, R. M., Song, J., and Poo, M. (1997). Propagation of activity-dependent synaptic depression in simple neural networks. *Nature*, 388:439–448.

Gawne, T. J., T.Kjaer, and Richmond, B. (1996). Latency: Another potential code for feature binding in striate cortex. *Journal of Neurophysiology*, 76(2):1356 – 1360.

Gerstner, W. (1991). Associative memory in a network of "biological" neurons. In *Advances in Neural Information Processing Systems*, volume 3, pages 84–90. Morgan Kaufmann, San Francisco.

Gerstner, W., Kempter, R., van Hemmen, L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383:76–78.

Gerstner, W., Ritz, R., and van Hemmen, J. L. (1992). A biologically motivated and analytically soluble model of collective oszillations in the cortex. *Biological Cybernetics*, 68:363–374.

Goodhill, G. J. and Sejnowski, T. J. (1997). A unifying objective function for topographic mappings. *Neural Computation*, 9:1291 – 1303.

Grossberg, S. (1976). Adaptive pattern classification and universal recording: II. feedback, expectation, olfaction, illusions. *Biological Cybernetics*, 23:187–202.

Haberly, L. (1985). Neuronal circuitry in olfactory cortex: anatomy and functional implications. *Chemical Senses*, 10(2):219–238.

Häfliger, P., Mahowald, M., and Watts, L. (1997). A spike based learning neuron in analog VLSI. In *Advances in Neural Processing Systems*, volume 9, pages 692–698. MIT Press, Cambridge (Mas.).

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing, New York.

Hebb, D. O. (1949). *Organization of Behavior*. Wiley, New York.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, 117:500–544.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to automata theory, languages and computation*. Addison-Wesley, Reading Mas.

Hopfield, J. J. (1995). Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376:33–36.

Johnston, D. and Wu, S. M. S. (1995). *Foundations of Cellular Neurophysiology*. MIT Press.

Katz, B. (1966). *Nerve, muscle, and synapse*. McGraw-Hill, New York.

Koch, C. and Bernander, O. (1995). Axonal modeling. In Arbib, M., editor, *Handbook of Brain Theory and Neural Networks*, pages 129–134. MIT Press, Cambridge.

Koch, C. and Segev, I., editors (1992). *Methods in Neural Modeling: From synapses to networks*. MIT Press, Cambridge.

Kohonen, T. (1993). Physiological interpretation of the self-organizing map algorithm. *Neural Networks*, 6:895 – 905.

Kohonen, T. (1995). *Self-Organizing Maps*. Springer, Berlin.

Krekelberg, B. and Taylor, J. G. (1997). Nitric oxide: what can it compute? *Network*, 8:1–16.

Leong, B. and Seiferas, J. (1981). New real-time simulations of multihead tape units. *J. of the ACM*, 28:166–180.

Leshno, M., Lin, V., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867.

Maass, W. (1995). Analog computations on networks of spiking neurons (extended abstract). In *Proc. of the 7th Italian Workshop on Neural Nets*.

Maass, W. (1996). Lower bounds for the computational power of networks of spiking neurons. *Neural Computation*, 8:1–40.

Maass, W. (1997a). Fast sigmoidal networks via spiking neurons. *Neural Computation*, 9:279–304.

Maass, W. (1997b). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10:1659–1671.

Maass, W. and Bishop, C., editors (1998). *Pulsed Neural Networks*. MIT Press (to appear).

Maass, W. and Natschläger, T. (1997). Networks of spiking neurons can emulate arbitrary hopfield nets in temporal coding. *Network*, 8(3):355–371.

Maass, W. and Ruf, B. (1995). On the relevance of the shape of postsynaptic potentials for the computational power of networks of spiking neurons. In *Proc. of the International Conference on Artificial Neural Networks ICANN*, pages 515–520, Paris. EC2.

Maass, W. and Ruf, B. (1998). On computation with pulses. *Submitted*.

Maass, W. and Zador, A. M. (1998). Dynamic stochastic synapses as computational units. In *Advances in neural processing systems (to appear)*. MIT Press, Cambridge.

Mainen, Z. F. and Sejnowski (1995). Reliability of spike timing in neocortical neurons. *Science*, 268:1503–1506.

Markram, H., Lübke, J., Frotscher, M., and Sakman, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275:213 – 215.

Markram, H. and Sakmann, B. (1995). Action potentials propagating back into dendrites triggers changes in efficacy of single-axon synapses between layer V pyramidal neurons. *Society for Neuroscience Abstracts*, 21:2007.

Markram, H. and Tsodyks, M. (1996). Redistribution of synaptic efficacy between neocortical pyramidal neurons. *Nature*, 382:807–810.

Murray, A. and Tarassenko, L. (1994). *Analogue Neural VLSI: A Pulse Stream Approach*. Chapman & Hall.

Natschläger, T. and Ruf, B. (1997a). Learning radial basis functions with spiking neurons using action potential timing. In *Neuromorphic systems: Engineering silicon from neurobiology (to appear)*. World Scientific.

Natschläger, T. and Ruf, B. (1997b). Spatial and temporal pattern analysis via spiking neurons. Technical report, Institute for Theoretical Computer Science, TU Graz. Electronically available at http://www.cis.tu-graz.ac.at/igi/bruf/.

Nicoll, R. A. and Malenka, R. C. (1997). Long-distance long-term depression. *Nature*, 388:427–428.

O'Keefe, J. and Reece, M. L. (1993). Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3):3317–30.

Rieke, F., Warland, D., Bialek, W., and de Ruyter van Steveninck, R. (1996). *SPIKES: Exploring the Neural Code*. MIT-Press, Cambridge.

Ritter, H. (1995). Self-organizing feature maps: Kohonen maps. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 846–851. MIT Press, Cambridge.

Ritter, H., Martinetz, T., and Schulten, K. (1992). *Neural Computation and Self-Organizing Maps*. Addison-Wesley, Reading, MA.

Roth, U., Jahnke, A., and Klar, H. (1997a). On-line hebbian learning for spiking neurons: architecture of the weight-unit of NESPINN. In *Proc. of the international conference on artificial neural networks (ICANN)*, pages 1217–1222. Springer Verlag, Berlin.

Roth, U., Jahnke, A., and Klar, H. (1997b). Simulation of spiking neural networks on different hardware platforms. In *Proc. of the international conference on artificial neural networks (ICANN)*, pages 1187–1192. Springer Verlag, Berlin.

Ruf, B. (1996). Pattern recognition with networks of spiking neurons. In *Workshop on Neural Networks Dynamics and Pattern Recognition, Toulouse*, pages 25–26. Onera, Centre d'Etudes et de Recherches de Toulouse.

Ruf, B. (1997). Computing functions with spiking neurons in temporal coding. In Mira, J., Moreno-Diaz, R., and Cabestany, J., editors, *Biological and artificial computation: From neuroscience to technology*, volume 1240 of *Lecture Notes in Computer Science*, pages 265–272. Springer, Berlin.

Ruf, B. and Schmitt, M. (1997a). Hebbian learning in networks of spiking neurons using temporal coding. In Mira, J., Moreno-Diaz, R., and Cabestany, J., editors, *Biological and artificial computation: From neuroscience to technology*, volume 1240 of *Lecture Notes in Computer Science*, pages 380–389. Springer, Berlin.

Ruf, B. and Schmitt, M. (1997b). Learning temporally encoded patterns in networks of spiking neurons. *Neural Processing Letters*, 5(1):9–18.

Ruf, B. and Schmitt, M. (1997c). Unsupervised learning in networks of spiking neurons using temporal coding. In Gerstner, W., Germond, A., Hasler, M., and Nicoud, J., editors, *Proc. of the 7th International Conference on Artificial Neural Networks*, volume 1327 of *Lecture Notes in Computer Science*, pages 361–366. Springer, Berlin.

Ruf, B. and Schmitt, M. (1998a). Self-organization of spiking neurons using action potential timing. *IEEE Transactions on Neural Networks (to appear)*.

Ruf, B. and Schmitt, M. (1998b). Self-organizing maps of spiking neurons using temporal coding. In Bower, J., editor, *Computational Neuroscience: Trends in Research 1998 (to appear)*. Plenum Press.

Sachs, L. (1984). *Applied Statistics: A Handbook of Techniques*. Springer, Berlin.

Segev, I. (1995). Dendritic processing. In Arbib, M. A., editor, *Handbook of Brain Theory and Neural Networks*, pages 282–289. MIT Press, Cambridge.

Segev, I., Rinzel, J., and Shepherd, G., editors (1995). *The theoretical foundation of dendritic function*. MIT Press, Cambridge.

Sejnowski, T. (1995). Time for a new neural code? *Nature*, 376:21 – 22.

Shepherd, G. M. (1994). *Neurobiology*. Oxford University Press.

Sirosh, J. and Miikkulainen, R. (1997). Topographic receptive fields and patterned lateral interaction in a self-organizing model of the primary visual cortex. *Neural Computation*, 9:577 – 594.

Stuart, G. J. and Sakmann, B. (1994). Active propagation of somatic action potentials into neocortical pyramidal cell dendrites. *Nature*, 367:69–72.

Tank, D. W. and Hopfield, J. J. (1987). Neural computation by concentrating information in time. *Proc. Natl. Acad. Sci. USA*, 84:1896–1900.

Thorpe, S., Fize, D., and Marlot, C. (1996). Speed of processing in the human visual system. *Nature*, 381:520–522.

Thorpe, S. and Gautrais, J. (1997). Rapid visual processing using spike asynchrony. In *Advances in Neural Information Processing Systems*, volume 9, pages 901–907. MIT Press, Cambridge.

Thorpe, S. and Imbert, M. (1996). Rapid visual processing using spike synchrony. In *Advances in neural processing systems*, volume 9, pages 901–907. MIT Press, Cambridge.

Thorpe, S. T. and Imbert, M. (1989). Biological constraints on connectionist modelling. In Pfeifer, R., Schreter, Z., Fogelman-Soulie, F., and Steels, L., editors, *Connectionism in Perspective*. Elsevier.

Tsodyks, M. and Markram, H. (1996). Plasticity of neocortical synapses enables transitions between rate and temporal coding. In von der Malsburg, C., von Seelen, W., Vorbrüggen, J. C., and Sendhoff, B., editors, *Proc. of the International Conference on Artificial Neural Networks*, pages 445–450. Springer, Berlin.

Tuckwell, H. (1988). *Introduction to Theoretical Neurobiology*. Cambridge University Press.

Wilson, M. and Bower, J. (1992). Cortical oscillations and temporal interactions in a computer simulation of piriform cortex. *J. of Neurophysiology*, 67:981–995.

Index