Ruhr-University Bochum

Master Thesis

Department of Electrical Engineering and Information Technology

Rate Coding and Temporal Coding in a Neural Network

Author:
Tim Utz Krause
108 009 235 231

Supervisor:
PD Dr. R. Würtz
Dipl. Phys. M. Leßmann

January 05, 2014

Declaration of authorship

Hiermit bestätige ich, dass ich die vorliegende Arbeit "Rate Coding and Temporal Coding in a Neural Network" selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Tim Utz Krause

Preamble

This thesis would not have been possible without support. Especially I want to thank Dr. Rolf Würtz. He developed the idea of this thesis and introduced me into the exciting research of Spiking-Neural-Networks. He was available at all times to answer any questions. His critical comments always helped me to keep the central theme for this work.

Furthermore I want to thank Markus Leßmann. He also was available for questions all times, helped me to write this thesis in the English language and spotted numerous mistakes.

Contents

ANN	Artificial-Neural-Network
FIR	Finite-Impulse-Response
SNN	Spiking-Neural-Network
STDP	Spike-Time-Dependent-Plasticity
PSTH	Peri-Stimulus-Time-Histogram
JPSTH	Joint-Peri-Stimulus-Time-Histogram
ISI	Inter-Spike-Interval
IF	Integrate-and-Fire
LIF	Leaky-Integrate-and-Fire
LTP	Long-Term-Potentiation
LTD	Long-Term-Depression
EPSP	Excitatory-Postsynaptic-Potential
IPSP	Inhibitory-Postsynaptic-Potential

Chapter 1

Introduction

Biological neural systems are certainly the most complex organs of living beings. Behaviour reaches from simple reflexes to complex cognitive capabilities like vision, speech or motion. Also emotions like sympathy, hate or love seem to have their origin in the neural system. The understanding of how the brain handles such complex tasks is fascinating and not yet well understood. Since the last decades are deemed to be the scientific epoch of genes and DNA, brain science is one of the essential scientific part of the upcoming epoch. From biological issues on cellular or even molecular level up to psychology on behavioural level it is a wide and interdisciplinary range. For a long time there is a desire to imitate biological nervous system in a bionic way. Not only the understanding of how the brain works is a goal, to create simple forms of artificial intelligence is in focus, too. Particularly with the popularity of the PC an interest has grown to equip it with cognitive skills.

Thus in the 1980s a 2nd generation of Artificial-Neural-Network (ANN)s were advanced, it finds applications in complex automatic control technique and other fields up to day. But there are some issues which cannot be solved with 2nd generation ANNs like the binding problem. Scientific research seems to be completed as far as possible. 2nd generation ANNs suffering from the problem that signals between neurons are modulated by simple scalar values. This is not corresponding to the biological ideal. It is assumed that information processing in the brain is based on electrical signals. These signals are very short impulses, called action potentials or spikes, running over nerve fibres and allow neurons to communicate with each other. A 3rd generation of ANNs, so called Spiking-Neural-Networks (SNNs) are trying to deal with this issue.

Great efforts have been made to study and unravel this spiking neural code, a general solution has not been found until today. Some methods to analyse spike trains of the biological ideal have been developed over the time. Findings in the spike trains led to different models which try to describe the information processing. Two basic principles are pointed out, rate coding and temporal coding. Section 2.1 of this thesis deals with spike train analysis and the signal processing in neural system.

Parallel to these investigations some partly very complex computer models have been developed. The aim is to proof theories and to get informations which cannot be measured in living organism. Another aim is the creation of artificial intelligence, as mentioned above. A simple neuron model, the Leaky-Integrate-and-Fire (LIF), is introduced in section 2.2. The simulation of SNNs differs from the simulation of 2nd generation ANNs, because of the temporal dimension. A new possibility to simulate aggregates of LIF neurons is introduced in section 2.3.

Chapter 3 deals with the possibilities to connect neurons with each other. The way neurons are connected has got an important effect on spike arrival times and how spikes are correlated. It is described in section 3.2. Further aspects of this chapter are neural circuit principle in feed forward networks (sec. 3.3) and recurrent networks (sec. 3.4). At the end of this chapter an introduction into synaptic plasticity and the associated ability to learn is given (sec. 3.5).

In chapter 4 a possible application of a SNN in the form of an image recognition is investigated. In a first instance some geometric shapes with large intersecting areas should be identified (sec. 4.2), in order to determine the influence of inhibiting connections. Afterwards it is the goal to recognise hand written digits from the MNIST database (sec. 4.3). Rate coding as well as time-to-first-spike coding is used.

The final chapter 5 abstracts the results and provides an outlook to further necessary investigations.

The appendix contains the results of some digit recognitions with different parameter settings (chap. A). Furthermore a brief summary of the classes of the programmed SNN software is given (chap. B).

Chapter 2

Theoretical background of a Spiking-Neural-Network (SNN)

The first section of this chapter gives an insight into neural coding schemes, furthermore an overview over some relevant analytical tools. Tools which are used to record and analyse data in the brain and are helpful in the study of artificial neural networks as well. After that the investigated neuron model is described in the following chapters and a new algorithm to simulate aggregates of these neuron models is introduced.

2.1 Signal processing in neural systems - neural coding

It is known that information transmission and communication of neurons take place by the use of action potentials. Action potentials are short electrical impulses, often also called spikes, running over the nerve fibres. Spikes are expected to be the underlying processes of information processing in the brain. The initiation and propagation of spikes and the underlying processes like depolarization, repolarization and hyperpolarization are important parts of research. But the main focus of the research of neural coding is to find out the reason of spike rates or even exact spike times of ensembles of neurons.

Caused by an interdisciplinary interest, broad efforts have been made over the years. Essential targets of the research are inter alia (Diesseroth, 2008):

- Biological prospecting
- Reverse engineering
- Medical translation

Curing diseases like parcinson or depression, building human-machine interfaces or creating cognitive and learning systems could be applications and benefits of this research. In summary the comprehension of the brain opens up nearly endless opportunities.

2.1.1 Spike train analysis methods

This subsection gives a brief introduction into common spike train analysis methods. Spikes are regarded as stereotype events here, represented by their spike time.

Further methods like *Spike pattern classification methods*, *Likelihood methods*, *Frequency-Domain methods*, *Neural spike train decoding* or the *Gravity transformation*, which deals with the lack of multiple spike train data analysis cannot be described in this thesis, even if these might be interesting for future investigations. Kirkland (2006) and Brown et al. (2004) are recommended to the interested reader.

Peri-Stimulus-Time-Histogram (PSTH)

This analysis method is based on the observation of a neuron while stimulating it with the same sequence for several repetitions. As a result one receives a set of spike trains. For small time windows Δt (typical one or a few milliseconds) the numbers of spikes $n_K(t; t + \Delta t)$ of all repetitions are summarized and are divided by the number of repetitions K. The spike density of the PSTH follows by scaling the result to the length of the time window Δt.

$$\rho(t) = \frac{1}{\Delta t} \frac{n_K(t; t + \Delta t)}{K} \tag{2.1}$$

This approach is very similar to the definition of a stochastic process, where the set of spike trains can be seen as realizations. The spike density of the PSTH corresponds to the first raw moment of the probability density function, which is given by evaluating ensemble averages.

Crosscorrelogram

The cross-correlation gives the possibility to compare two different neurons by giving a measure of the similarity of their spike trains. One neuron is chosen as reference cell, the other as target cell. For each spike of the reference spike train, the existing time lag to each spike of the target spike train is determined. Subsequently the number of time lags falling into equi-sized intervals of time, called bins, are counted. One receives something like a histogram of time lags, whereby as well positive as negative delays are listed. If reference and target neurons are swapped, one receives the same cross-correlation function, only swapped in time too. A more detailed description and some caveats are given in Kirkland (2006).

Joint-Peri-Stimulus-Time-Histogram (JPSTH)

The JPSTH is another cross-correlation analysis method, a two dimensional histogram of joint spike counts. Each axis of the two dimensional histogram represents the temporal observation of one neuron, divided into small time windows. A bin $(x; y)$ is incremented if neuron 1 fires at time x and neuron 2 at time y. In this way all spikes of the two neurons are brought together. Similar to the PSTH, spikes are recorded over several stimulus trials. Compared with the crosscorrelogram the advantage of the JPSTH lies mainly in the possibility to get a temporal relation to the stimulus.

The main diagonal of the JPSTH is of particular importance. It displays the rate of simultaneous or almost simultaneous firings over time. Sometimes it is plotted in a separate diagram and is called coincidence histogram then.

2.1.2 Temporal coding and rate coding

There have been great efforts in recent years to unravel the neural code, but no general solution has been found yet. Two of the most common principles are called temporal coding and rate coding, although there is no broad definition of them in literature. Some approaches are described in Gerstner et al. (2008) and Thorpe et al. (2001). A selection is summarized in the following.

Another, not less interesting, question is how information is encoded in sensory cells in response to stimuli like light, sound, temperature, mechanical stimulation, chemical stimulation, etc. and how the neural code is decoded for example in muscles. Unfortunately this cannot be discussed here.

Rate coding

Assuming neural systems use rate codes to exchange information, the firing rate contains all of the information. Rate codes are very tolerant to disturbances, but have a low information density.

Rate coding could be observed in sensory and motor systems early. One of the best known examples is the firing rate of a stretch receptor in a muscle spindle (Adrian et al., 1926). In general the approach of temporal average works well for a slowly varying or constant stimulus. It has fundamentally led to the second generation of artificial neural networks, whose information exchange and processing is based on one scalar value per neuron, the activation ν.

Differentiations in rate coding are made as follows:

- *Spike count rate - average over time*
  Assuming the spike count rate as basis of coding in neural systems, the rate a neuron fires spikes carries all the information. Measuring the spike count rate can merely be done by counting the spikes in a time interval Δt and dividing it by Δt, what means calculating the temporal average. The length of the time interval depends on the type of neuron and the stimulus (typical one hundred or a few hundred milliseconds).

- *Rate as spike density - average over several runs*
  This coding scheme can be seen analogously to analysing with the PSTH. It is sufficient for measuring and evaluating neural activity, but it is not suitable to describe signal processing in the brain. The example of a frog which wants to catch a fly is given in Gerstner et al. (2008). The frog cannot wait since the fly flies along exactly the same trajectory several times. The frog has only got a single chance.

- *Rate as population activity - average over several neurons*
  This coding scheme is based on the idealization of populations of neurons with identical properties. Populations of neurons with similar properties can be found for instance in the visual cortex of cats and monkeys. The number of neurons concurrently active in a population is relevant.

Temporal coding

Regarding temporal coding, the information is contained in the exact moments of spike occurrence. In contrast to rate coding, temporal coding enables a higher level of information density and a higher speed of processing because each spike counts and the determination of average is not necessary. However, minor disturbances may have major effects.

S. Thorpe did some research on the speed of processing in the human visual system (Thorpe et al., 1996). He analysed the time needed to recognize animals on photographs, which can be done in less than 150 ms. Considering the number of processing stages involved within the brain, he came to the conclusion "that much of this processing must be based on essentially feed-forward mechanisms". In such a short period of time an determination of average over several spikes is not possible. This is an indication that the first spikes probably contain most of the information.

Differentiations in temporal coding are made as follows:

- *Count coding and binary coding*
  A post-synaptic neuron is stimulated by n pre-synaptic neurons, whereby each pre-synaptic neuron should fire either once or not at all within the regarded time window after the stimulus. The simplest choice would be counting the number of pre-synaptic neurons emitted a spike. $n + 1$ states of the system are possible. A more efficient way is using the neurons as a binary code, then 2^n states are possible.

- *Rank Order Coding*
  When using the order in which the neurons fire, one talks about rank order coding. If one considers one or none firing per neuron again and no simultaneous firing, $n!$ possibilities exist.

- *Correlation and synchrony*
  The principle of rank order coding can be extended by regarding the exact Inter-Spike-Interval (ISI). Firstly synchrony between two or more spikes could be seen as an event. But it is also possible to consult correlations between pairs or more pre-synaptic spikes, thus any precise spike pattern could be an event.

- *Time-to-first-spike*
  This coding assigns importance to the precise timing of the very first spikes after a neural network is stimulated. The strength of stimulation is coded into the time-to-first-spike, triggered by an external stimulus. The stronger a stimulus, the earlier the spike. In an idealized network each neuron only fires once, after that it is calm until the next stimulus of the network occurs.

- *Phase coding*
  Phase coding requires a reference oscillation (in many brain areas background oscillations are usual). The spike of a neuron is able to carry information via the phase to the reference oscillation. This is very similar to the time-to-first-spike code, with the sole difference that the trigger is the reference oscillation instead of the stimulus.

Comparison of rate coding and temporal coding

When measuring the membrane potential of a nerve fibre over time, a continuous aperiodic signal is obtained. Each continuous aperiodic signal can be transformed from time domain to frequency domain by Fourier-transformation. It is a transformation to another mathematical space merely, the contained information remains unaltered. Thus where exactly is the difference between the concepts of temporal coding and rate coding?

Both coding schemes are apparently used at the same time and the same neural systems and cannot be clearly distinguished from one another. As well as this rate-time duality, the presented varieties of rate and temporal coding are not valid or obligating on their own. Furthermore a clear demarcation between rate and temporal code will not always be possible. Therefore it would be more appropriate to understand the introduced coding schemes as different descriptions of a black box system. They are not suitable as general interpretations of principles of neural coding.

2.2 (Leaky-)Integrate-And-Fire-Model

With the invention of the electron microscope in the 1950s the neuron doctrine could be proofed. Neuronal systems are made up of individual cells, the neurons. These are the elementary processing units and they connected to each other by electrical or chemical synapses. The exchange of information take place by short electrical impulses, called spikes or action potentials. It is not unusual that one neuron in the vertebrate cortex addresses ten thousand (10^4) post synaptic neurons. In addition to hundred billion neurons (10^{11}), the human brain consists of a trillion (10^{12}) glia cells. Glia cells are seen as "supporter" cells for structural stabilization and energy supply, but their role and functionality in respect to information processing is largely unknown. Attending to the individual behaviour of each neuron, which has to be studied on a cellular or even molecular level, the functionality of such large circuits is rather complex and part of current and future research of neurobiology. The detailed biological background cannot be described sufficiently and will not be continued further at this point. This can be found in literature like Bear et al. (2008).

Several neuron models have been developed over time. They differ in biological plausibility and computational efficiency. Several models are listed in Izhikevich (2004) in a comparative way. One of the most plausible and detailed model has been performed by Hodgkin and Huxley (1952). This conductance based model is composed of a set of differential equations which describe the membrane potential caused by the activation of Na and K and the inactivation of Na current flows. Over the time the model has been developed further. Newer models include more types of ion channels and take account of the neurons and synapses geometry (Gerstner et al., 2008). The extremely complex calculation is a drawback of the *Hodgkin-Huxley-Model.* Only single neurons or small populations of neurons can be simulated within an acceptable length of time (Izhikevich, 2004). Therefore also efforts have been made to simplify the *Hodgkin-Huxley-Model.* A model which only needs three differential equations is the *Hindmarsh-Rose-Model.* Models with two differential equations are for example the *Morris-Lecar-Model*, the *Fitz-Hugh-Nagumo-Model* and the *Izhikevich-Model.* Besides offering an opportunity to speed up calculation, two dimensional models can be studied in the phase plane. Two general types (type I and type II) can be distinguished by their fixed points in the phase plane.

For the studies of neural coding, memory, and network dynamics even more simple phenomenological neuron models are used. Some of the best known are the *Leaky-Integrate-and-Fire (LIF)* and its derivatives and the *Spike-Response-Model.* Unlike in the above mentioned conductance-based neuron models, spikes in formal spiking neuron models are characterized by their firing time. The LIF neuron model is used for the studies in this thesis. It will be described in detail in the following section.

2.2.1 The basic circuit

This subsection has been essentially inspired from chapter 4.1.1 of Gerstner et al. (2008).

Threshold conditions, reset value and absolute refractory period

In formal neuron models spikes are regarded as stereotyped events. When a spike reached a synapse, a Excitatory-Postsynaptic-Potential (EPSP) increases the membrane potential of the post-synaptic neuron. If the membrane potential $u(t)$ reaches an upper threshold ϑ_{u} at this moment, the neuron sends out a spike itself. The fth spike event occurs at the time of threshold crossing $t^{(f)}$.

$$t^{(f)}: \qquad u(t^{(f)}) = \vartheta_{\mathrm{u}} \tag{2.2}$$

It is supposed that through an action potential the capacitor discharges directly, which is in fact almost practically impossible. This leads to the membrane potential being reset to the value u_{r}, where $u_{\mathrm{r}} < \vartheta_{\mathrm{u}}$.

$$\lim_{t \to t^{(f)}} u(t) = u_{\mathrm{r}}, \quad t > t^{(f)} \tag{2.3}$$

Next to processes which are increasing the post-synaptic neurons membrane

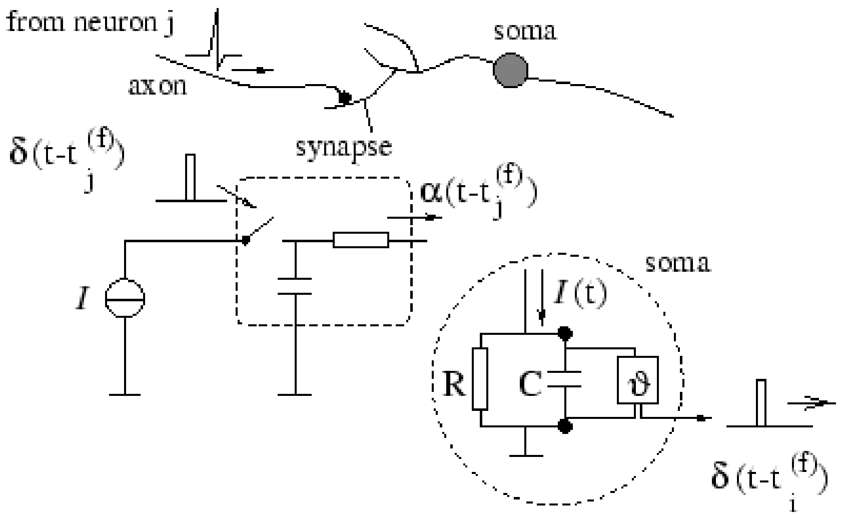


Figure 2.1: Schematic diagram of the *Integrate-And-Fire-Model*. The basic circuit is the module inside the dashed circle on the right-hand side. A current $I(t)$ charges the RC circuit. The voltage $u(t)$ across the capacitance (points) is compared to the upper threshold ϑ_{u}. If $u(t) = \vartheta_{\mathrm{u}}$ at time $t_i^{(f)}$ an output pulse $\delta(t - t_i^{(f)})$ is generated. Left part: A pre-synaptic spike $\delta(t - t_j^{(f)})$ is low-pass filtered at the synapse and generates an input current pulse $\alpha(t - t_j^{(f)})$. (Figure with caption taken from Gerstner et al. (2008), Fig. 4.1)

potential, processes which are decreasing the post-synaptic neurons' membrane potential do exist. This inhibition of a neuron is called Inhibitory-Postsynaptic-Potential (IPSP). In fact the biochemical process of inhibition is much more complex, but it is substituted to a negative sign here. If neural inhibition is supposed to be utilized, a lower threshold ϑ_{l} has to be considered as well. Inhibitions are bounded to this threshold thus lower membrane potentials are not possible.

Consequently the membrane potential is always within the interval $\vartheta_{\mathrm{l}} \leq u_{\mathrm{m}} < \vartheta_{\mathrm{u}}$. Likewise the reset value u_{r} must be within this interval.

$$\vartheta_{\mathrm{l}} \leq u_{\mathrm{r}} < \vartheta_{\mathrm{u}} \tag{2.4}$$

After a neuron initiated an action potential, it takes a certain time until the neuron is able to generate a second signal. The reason for this is that the action potential goes along with a depolarization of the membrane and it takes some time to repolarise it. Until the membrane is not repolarised, no further depolarisation is possible. Thus the neuron is blind to incoming spikes within this period of time, which is called absolute refractory period Δ_{abs}.

Integrate-and-Fire (IF)

As the name already indicates, the main element of the IF model is an integrator. The electrical component is a capacitor C_{m}. This resembles the biological structure of a cell. Two conductive layers, the inter-cellular fluid and the interstitium, which are separated by an dielectricum, the cell membrane. The charging current into the capacitor i_{IF} is given by the following equation.

$$i_{\mathrm{IF}}(t) = C_{\mathrm{m}} \, \frac{\mathrm{d}u_{\mathrm{IF}}}{\mathrm{d}t} \tag{2.5}$$

And the membrane potential is given by

$$u_{\mathrm{IF}}(t) = u_{\mathrm{r}} + \frac{1}{C_{\mathrm{m}}} \int_0^{t-\hat{t}} i_{\mathrm{IF}}(t-s)\; ds \; . \tag{2.6}$$

$\hat{t}$ is the time of the latest spike occurrence.

Leaky-Integrate-and-Fire (LIF)

The main element of the IF model is an integrator embodied by a capacitor. In addition the LIF neuron model considers that the membrane has got an electrical conductivity $1/R_\mathrm{m}$. This means that the membrane is leaking and therefore the membrane potential decreases over time. In the electrical circuit both constructional elements are parallel connected, see Fig. 2.1. The incoming current $i_\mathrm{LIF}(t)$ splits into a dissipation current through the resistor $i_{\mathrm{R_m}} = u_\mathrm{LIF}/R_\mathrm{m}$ and a current which is charging the capacitor $i_{\mathrm{C_m}} = C_\mathrm{m}\,\mathrm{d}u_\mathrm{LIF}/\,\mathrm{d}t$. The following linear differential equation of first order is obtained.

$$i_\mathrm{LIF}(t) = \frac{u_\mathrm{LIF}(t)}{R_\mathrm{m}} + C_\mathrm{m}\,\frac{\mathrm{d}u_\mathrm{LIF}}{\mathrm{d}t} \tag{2.7}$$

This equation can be converted into the standard form by multiplying with R_m. The resulting product $R_\mathrm{m}C_\mathrm{m}$ equates the time constant of the RC circuit τ_m.

$$\tau_\mathrm{m}\,\frac{\mathrm{d}u_\mathrm{LIF}}{\mathrm{d}t} = R_\mathrm{m}C_\mathrm{m}\frac{\mathrm{d}u_\mathrm{LIF}}{\mathrm{d}t} = -u_\mathrm{LIF}(t) + R_\mathrm{m}\;i_\mathrm{LIF}(t) \tag{2.8}$$

A solution of this differential equation under the initial condition u_r can be found by integration. $\hat{t}$ is the time of the latest spike occurrence.

$$u_\mathrm{LIF}(t) = u_\mathrm{r}\;\exp\left(-\frac{t-\hat{t}}{\tau_\mathrm{m}}\right) + \frac{1}{C_\mathrm{m}}\int_0^{t-\hat{t}} \exp\left(-\frac{s}{\tau_\mathrm{m}}\right) i_\mathrm{LIF}(t-s)\;ds \tag{2.9}$$

Stimulation with a constant current

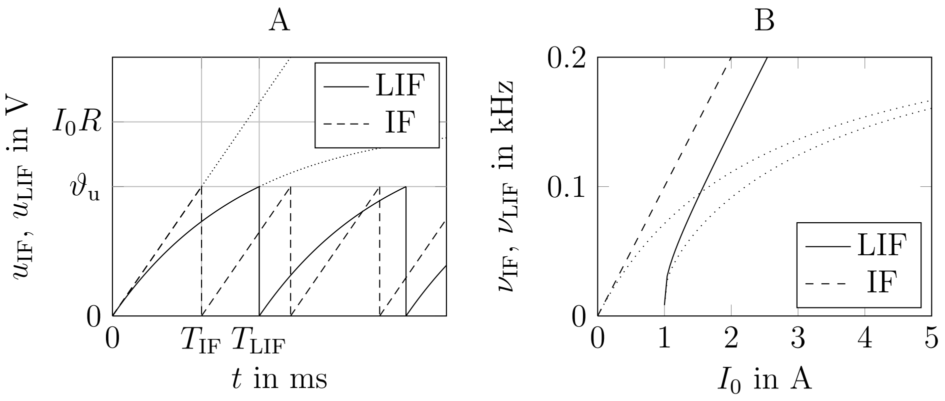


Figure 2.2: Constant stimulation of a IF neuron with reset potential u_r = 0V, capacity C_m = 0.01F and in the case of LIF a resistance R_m = 1Ω. **A:** Membrane potential as function of time, input current I_0 = 1.5A. **B:** Resulting firing rates as a function of constant stimulation currents, without (solid/dashed lines) and with (dotted lines) absolute refractory period Δ_abs = 4ms

To distinguish between the fundamental principles, both models are stimulated with a constant input current $\alpha(t) = I_0$ in a first step. Assume the membrane potential is taken to $u_\mathrm{r} = 0$ and the latest spike occurred at $\hat{t} = 0$. This procedure can be considered as the step response of the neuron models.

The membrane potential trajectory of the IF neuron equals the charge stored in the capacitor related to its capacity. The charge flowing toward the capacitor can be calculated easily by integrating the input current over time. Thus the membrane potential rises linearly with time, until the upper threshold is reached. At that time the membrane potential is reset and the charging process restarts.

$$u_\mathrm{IF}(t) = \frac{1}{C_\mathrm{m}} \int_{\hat{t}}^{t} I_0 \; dt = \frac{1}{C_\mathrm{m}} \; I_0 \; (t - \hat{t}) \tag{2.10}$$

The membrane potential of the LIF can be calculated by the following equation.

$$u_\mathrm{LIF}(t) = R_\mathrm{m} I_0 \left[1 - \exp\left(-\frac{t - \hat{t}}{\tau_\mathrm{m}} \right) \right] \tag{2.11}$$

Without threshold the membrane potential would always become $u_\mathrm{LIF}(\infty) = R_\mathrm{m} I_0$ for $t \to \infty$. Thus a LIF neuron will never fire while $I_0 R_\mathrm{m} < \vartheta_\mathrm{u}$.

At each time the membrane potential resets the neuron fires a spike. The time interval between two spikes can be calculated by equating equation 2.10 and 2.11 with the threshold potential ϑ_u and rearranging the resulting equation.

$$T_\mathrm{IF} = \Delta_\mathrm{abs} + \vartheta_\mathrm{u} \frac{C_\mathrm{m}}{I_0} \tag{2.12}$$

$$T_\mathrm{LIF} = \Delta_\mathrm{abs} + \tau_\mathrm{m} \ln \left(\frac{R_\mathrm{m} I_0}{R_\mathrm{m} I_0 - \vartheta_\mathrm{u}} \right) \tag{2.13}$$

The frequencies the neurons fire with are given by $\nu_\mathrm{IF} = 1/T_\mathrm{IF}$ and $\nu_\mathrm{LIF} = 1/T_\mathrm{LIF}$. Firing rates of neurons without absolute refractory period are not bounded. The theoretical maximum firing rate for a neuron with absolute refractory period is $\nu_\mathrm{IF} = \nu_\mathrm{LIF} = 1/\Delta_{abs}$. But this value is never reached because it will always take a very small but finite time to generate an action potential after the refractory period is expired.

Stimulation with spikes

Stimulating the neuron models with constant currents is not very authentic. As mentioned above spikes are seen as stereotype events. Arbitrary pulse shapes $\alpha(s)$ are in principle possible. Besides sophisticated but also more complex functions, a Dirac-delta function is the simplest version which can

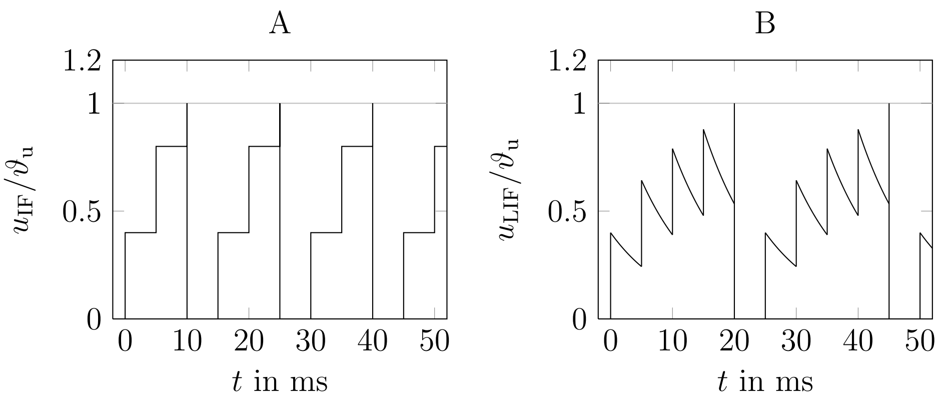


Figure 2.3: Membrane potential normalized to the upper threshold ν_{u} of **A:** an IF neuron ($C_{\mathrm{m}} = 10\mathrm{mF}$) and **B:** a LIF neuron ($R_{\mathrm{m}} = 1\mathrm{k}\Omega$, $C_{\mathrm{m}} = 10\mathrm{mF}$) during stimulation with a continuous firing rate $\nu = 0.2\mathrm{kHz}$ and a loading of $q_{\mathrm{s}} = 4\mathrm{mC}$ per spike.

be chosen. A Dirac-impulse $\delta(s)$ can be imaginated as an infinitely high and infinitely thin impulse at a time s, which is of course physically infeasible. The integral over a single Dirac-impulse is defined as $\int \delta(s)\ \mathrm{d}\,s = 1$. Assuming that each spike emitted by a neuron carries the electrical charge q_{s}, the arbitrary pulse shape $\alpha(s)$ is given by

$$\alpha(t - t^{(f)}) = q_{\mathrm{s}}\ \delta(t - t^{(f)})\ . \tag{2.14}$$

The impulse response of an IF-neuron is

$$h_{\mathrm{IF}}(t) = \frac{Q_0}{C_{\mathrm{m}}} + \frac{q_{\mathrm{s}}}{C_{\mathrm{m}}}\ , \tag{2.15}$$

whereby the first term represents the electrical charge which is already stored in the capacitor. Merely the electrical charge of the incoming spike must be added.

For the calculation of the impulse response of a LIF neuron, one considers that no electrical current flows through the resistor during the infinitesimal duration of the incoming spike. Then the charge increases due to the incoming spike. Immediately after the spike event the capacitor begins to discharge over the resistor. The discharge process is given by an exponential decay with the time constant $\tau_{\mathrm{m}} = R_{\mathrm{m}}C_{\mathrm{m}}$.

$$h_{\mathrm{LIF}}(t) = \left(\frac{Q_0}{C_{\mathrm{m}}} + \frac{q_{\mathrm{s}}}{C_{\mathrm{m}}}\right) \exp\left(-\frac{t}{\tau_{\mathrm{m}}}\right) \tag{2.16}$$

Artefacts caused by quantization of time base

The quantization of time base causes some artefacts because there is a possibility that several spikes reach a neuron at the same time. Equality of spike times can be caused by user defined scenarios with whole-numbered spike times or from accidentally simultaneously arriving spikes. This could play a critical role if an EPSP and an IPSP reach a neuron simultaneously. The neuron may not fire if the IPSP is processed first because the membrane potential is decreased as much as the EPSP is not sufficient to evoke the action potential. The other way around the EPSP may causes an action potential before the IPSP is processed. To deal with this problem, spikes with equal times are summed up before they are processed.

In this thesis a floating point format is used as temporal base, whereby the comparison of floating point variables is not a trivial problem. Here two values are seen as equal if their difference is below a threshold ϵ. Thus a small temporal error in the range of ϵ remains. If these spikes evoke an action potential of the receiving neuron, this small error relays. A further remaining problem is that spikes could be suppressed by the time shifting with ϵ. If there is no refractory period and activities of the involved spikes are very large, the activities of the summed spikes may be strong enough to be able to cause more than one spike. Through the combination the neuron only fires once.

2.2.2 Spatial and temporal summation

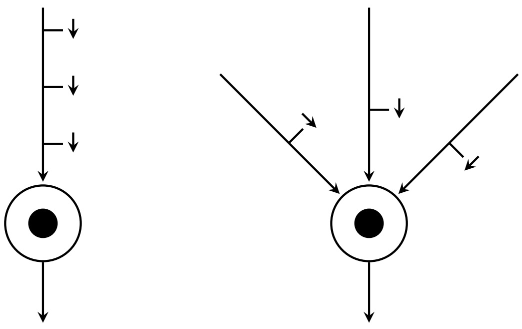


Figure 2.4: Temporal summation (left) and spatial summation (right) of spikes running towards a neuron.

A single pre-synaptic spike is normally not sufficient to evoke a spike event in the post-synaptic neuron. This is because each spike carries only a little charge which is not able to modify the post-synaptic neurons membrane potential in the degree needed for evoking a spike event. Thus a superposition of several spikes is necessary to be successful. Spikes can reach a neuron spatially separated by different synapses or temporally separated one by one over one synapse. The resulting input charge of a post-synaptic neuron i, caused by spatial and temporal summation of spikes of pre-synaptic neurons j, can be mathematically expressed by the following equation.

$$Q_i(t) = q_\mathrm{s} \underbrace{\sum_j w_{i,j} \underbrace{\sum_f \delta(t - t_i^{(f)} - d_{i,j})}_{\text{temporal summation}}}_{\text{spatial summation}} \tag{2.17}$$

The temporal term $d_{i,j}$ represents the delay a spike experiences while propagating along the axon and dendrites from a pre-synaptic neuron j to a post-synaptic neuron i. Each spike carries the identical charge q_s. The inner sum represents the superposition of f consecutive spikes, transmitted by a single connection. To allow different efficacies of synapses, each connection between a spike transmitting neuron j and a receiving neuron i is weighted by an individual factor $w_{i,j}$. These factors are taken account within the outer sum which represents the spatial summation. All weight factors can either be positive for an excitatory connection or negative for an inhibitory connection, but cannot change their signs over the time. Delays are positive of course.

To allow more than one connection between two neurons i and j, the sum has to be extended by a further variable k which counts the several connections.

$$Q_i(t) = q_\mathrm{s} \sum_{i,k} w_{i,j,k} \sum_f \delta(t - t_i^{(f)} - d_{i,j,k}) \tag{2.18}$$

These equations represent the total input charge of a neuron i which differs from the current charge in the neurons capacitor. To calculate the current membrane potential, it must be taken care of the temporal behaviour of the neuron model, see 2.2.1.

2.2.3 Neuron and connection parameters

All previously introduced electrical parameters can be superseded by dimensionless parameters to obtain a more phenomenological model. The time base could also be superseded by a generic unit of time, but is maintained in milliseconds to keep compliance to biological models.

For reasons of simplicity the reset value is set to zero ($u_\mathrm{r} = 0$), the upper threshold is set to one ($\vartheta_\mathrm{u} = 1$), the lower threshold is set to zero ($\vartheta_\mathrm{l} = 0$) and the electrical charge of each spike is set to one $q_\mathrm{s} = 1$ from here on, unless otherwise mentioned. The capacitor C_m and resistor R_m are neglected. C_m can be depicted as scaling factor of the membrane potential (multiplying 2.6 respectively 2.9 with C_m). Thus the factor C_m can be set to the fix value $C_\mathrm{m} = 1$ and is represented by the threshold values ϑ_u and ϑ_l. The time constant of the electrical circuit τ_m is given by the product $R_\mathrm{m} C_\mathrm{m}$. With a given C_m, R_m is a simple proportionality factor ($R_\mathrm{m} \propto \tau_\mathrm{m}$). Thus the membrane resistor can be set to the fix value $R_\mathrm{m} = 1$ and is represented by the membrane time constant τ_m.

In conclusion, if $\mathcal{N}$ is a set of neurons, each neuron $n \in \mathcal{N}$ can be described by the following variables:

- Upper threshold: $\vartheta_\mathrm{u} > 0$
- Lower threshold: $\vartheta_\mathrm{l} \leq 0$
- Absolute refractory period: $\Delta_\mathrm{abs} \geq 0$
- Membrane (leaky) time constant: $\tau_\mathrm{m} > 0$

$\mathcal{C}_{i,j,k}$ is a set of connections between pre-synaptic neurons $j \in \mathcal{N}$ and post-synaptic neurons $i \in \mathcal{N}$. k represents the kth connection between two neurons. Each connection $c \in \mathcal{C}$ can be described by the two following variables:

- Delay: $d_{i,j,k} > 0$
- Weight: $w_{i,j,k} \begin{cases} \geq 0, & \text{for excitatory synapses} \\ \leq 0, & \text{for inhibitory synpses} \end{cases}$

Delays $d_{i,j,k}$ are always positive, if the connection is excitatory $w_{i,j,k} \geq 0$ and $w_{i,j,k} \leq 0$ if the connection is inhibitory. If there is no usage of several connections between two neurons, k is not listed from here on.

2.3 Simulation flow and multi-threading

One goal of this thesis is a simple model to simulate aggregates of neurons in a moderate time on a commercial personal computer. There are several possibilities to calculate such networks.

One way could be a continuous simulation with a time discretisation. This means one has to calculate the activity of each neuron for each time

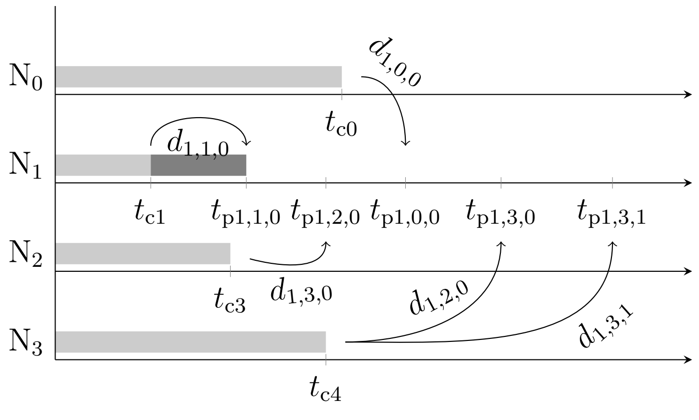


Figure 2.5: Neuron n_1 aims to be calculated, the question is how far its input is predictable. The four predecessors n_0, n_1 (output looped to input), n_2 and n_3 are calculated until t_{ci} (light gray). If one of them delivers a new spike, this spike can reach neuron n_1 at $t_{p1,i,0} = t_{ci} + d_{1,i,0}$ first. The minimum is given by the loop connection $t_{p1,1,0} = t_{c1} + d_{1,1,0}$, no new spike of the predecessors can reach neuron n_1 prior, therefore it can be calculated until $t_{p1,1,0}$ (dark gray).

step. As depicted above, action potentials are seen as stereotype events and are mathematically described as Dirac-impulses here. This makes it possible to characterise each action potential with its infinitesimal duration in time scale by a time mark. A continuous simulation using a time discretisation with fixed or variable time steps as well, would not be expedient and not even possible. Looking at one neuron's time course, nothing happen in the majority of time. The majority of simulation time would be paid to do nothing. Another reason is that the sample frequency would have to be selected very high to achieve sufficient precision. The Nyquist-theorem $T < 1/(2f_g)$ could never be fulfilled because the Fourier-transformation is $\mathcal{F}(\delta(t)) = 1$ and therefore $f_g \to \infty$.

Another way is demonstrated in Mayerhofer et al. (2002), where discrete event simulation (DES or DEVS) is used. Each spike reaching a neuron counts as an event, all events are executed sequentially.

In this thesis a new and very performant strategy is used. It offers high computational power and raises the possibility of multi-threading. For easy understanding of this strategy, a simple example is discussed first. In figure 2.5 one sees the progress of the calculation of four neurons. The calculation of neuron n_1 is less proceeded than the calculation of the neurons n_0, n_2 and n_3. Neuron n_1 shall be simulated next. Because each connection has got a delay, each spike has got a death time while propagating toward a neuron. The first further spike of neuron n_0 could reach neuron n_1 at $t_{\mathrm{p}1,0,0} = t_{\mathrm{c}0} + d_{0,1,0}$, the first further spike by itself (output looped to input) at $t_{\mathrm{p}1,1,0} = t_{\mathrm{c}1} + d_{1,1,0}$ and so on. If all predecessors are considered, the minimum of $t_{\mathrm{p}i,1,k} = t_{\mathrm{c}i} + d_{1,i,k}$ depicts in which range no further input spikes will occur and how far neuron n_1 can be calculated. For this strategy a positive definite delay matrix is necessary. Negative entries or entries equal to zero would lead to a situation, where the input of a neuron is not predictable and therefore not possible to calculate with this strategy.

A major advantage of this strategy is, that the determination of predictable input spikes works for all neurons all over the time. This means, that not only the neuron with the least simulation progress can be calculated. In principle it would be possible to start an own thread for each neuron, which first calculates the maximal predictable time, and than simulates the neuron inputs within this time and repeats these steps again and again. In practice a smaller number of threads would be suitable.

Another advantage is that in pure feed forward networks each neuron has to be called only once, under the condition that they are called in order of their arrangement, neurons of the input layer first, then hidden layer neurons and at last the output layer neurons. This leads to a rather fast simulation progress.

If only the neuron with the least simulation progress is calculated at all times (a single thread) and there are large numbers of connections, a simplification is possible. Instead of considering the exact calculation time $t_{\mathrm{c}i}$, one just has to ensure that all pre-synaptic (spike emitting) neurons are processed as far as the post-synaptic (resolving) neuron. Then no spike can arrive at the post-synaptic neuron's soma within the minimal afferent delay. Concerning just the minimal afferent delay might be slightly shorter than concerning the processed time of the pre-synaptic neuron, too. But it has got the great advantage that the minimal afferent delay does not change over time, and therefore can be determined once.

Chapter 3

Fundamental investigations and neural circuit principles

While previous chapters were concerned with the description of the here used neuron model, it is the goal of this chapter to examine the interaction between neurons. The applicability to more complex tasks is not yet in focus and follows with the example of recognition of hand written digits in chapter 4. Here the aim is to examine intrinsic and in scope restricted circuits to get a better understanding of processes and signal sequences within SNNs. Or in other words, to get a better understanding of neural coding. Furthermore it improves comprehension of the factors of influence and helps to parametrise more complex networks. Simple connections between neurons are studied first. Afterwards the importance of correlation and synchrony in spike trains is pointed out. Feed forward networks are briefly discussed in the following. After that some examples of recurrent circuits are analysed. Finally possibilities of training SNNs are described, whereby the main focus is on STDP.

3.1 Divergence and convergence

Regarding a soma of a single neuron, two generally conduction are possible, inputs and outputs. Neural outputs corresponding to the neurons axons and the inputs to its dendrites (Bear et al., 2008). The process of spreading a neural activity to several outputs is called divergence, see fig. 3.1 (left). Caused by the underlying bio-electrical process of signal transmission, the signal intensity is not decreased by an increasing number of paths. The superposition of several inputs is called convergence, see fig. 3.1 (right).

A general problem occurs if spikes, converging toward a neuron $\mathrm{N_i}$, carry

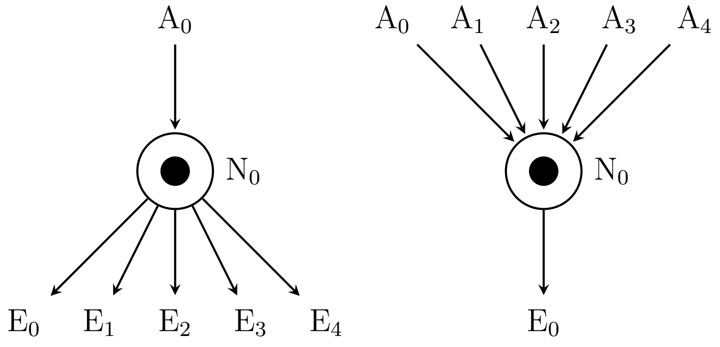


Figure 3.1: Divergence (left) of one input spike train A_0 to several outputs $E_0 \ldots E_4$ and convergence (right) of several input spike trains $A_0 \ldots A_4$ to one output spike train E_0.

large charges q_s or if the weights $w_{i,j}$ are very high. For the sake of simplicity a simple IF neuron without any refractory time Δ_abs is considered here. If $w_{i,j} \geq \vartheta_\mathrm{u}$ each incoming spike leads to an output spike, thus the input firing rate equals the output firing rate and weight changes have no effect. This contradicts the assumption that weight changes are responsible for learning.

For smaller weights $w_{i,j} \leq \vartheta_\mathrm{u}$ the cell sums up the incoming activities as integrator, more than one spike is required to evoke an output spike. In fig. 3.2 the number of required incoming spikes needed to effect an action potential of a single neuron is displayed over various weights. The formalism behind is:

$$F = \mathrm{ceil}\left(\frac{1}{w}\right),\ 0 < w_{i,j} \leq 1 \tag{3.1}$$

Because the number of required spikes goes with $1/w$ the influence of small weights is very small.

If all spikes are transmitted by a single connection, the neuron works as a frequency divider which can be adjusted by the weight:

$$\nu_\mathrm{out} = \frac{\nu_\mathrm{in}}{\mathrm{ceil}\left(\frac{1}{w}\right)},\ 0 < w_{i,j} \leq 1 \tag{3.2}$$

The smaller the weights, the finer the adjustability.

The exact timing of the latest input spike, which evokes the output spike, might play a role, too. The time to the first output spike would be $1/\nu_\mathrm{out}$ for example. Here, too, a finer tuning and higher precision can be attained by smaller weights.

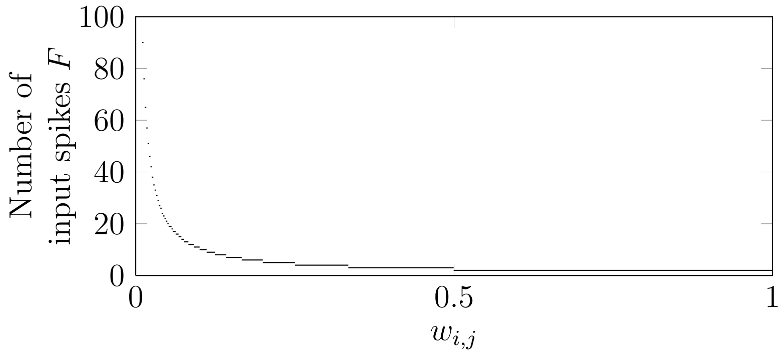


Figure 3.2: Number of required input spikes F needed to effect one output spike of a single neuron $\mathrm{N_i}$, displayed over various weights $w_{i,j}$. Concerning an IF neuron without refractory period Δ_{abs}.

3.2 Correlation and synchrony

The behaviour described in the previous section is caused by the neural integration inside the IF. The temporal dynamics being left out from the influences so far. These are described in this section and will ultimately lead to the dualism of the role of a neuron as integrator or coincidence detector.

3.2.1 A neuron as bandpass filter

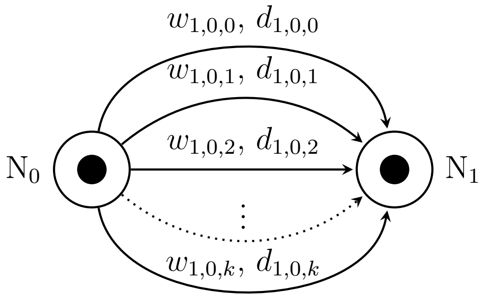


Figure 3.3: Two neurons with several connections.

Looking at two neurons and the possibility to connect these two neurons by multiple paths, it reminds strongly of the structure of a Finite-Impulse-Response (FIR) filter. Taking a FIR filter the incoming signal goes through a tapped delay line, whereby a different weighting is used for each time delay. If one considers a neuron instead of the FIR, differences in delays arises as a result of the length or myelinization of the nerves in the different paths. Differences of weights are given by different strengths of synapses. This structure provides the possibility to design low pass, high pass and band pass filter characteristics by varying the delays and weights. Its impulse response is always finite and therefore the filter is always stable. Whole artificial neural networks can be designed by modelling connections between neurons by FIR filters. This technique enhances the classical perceptron architecture to a temporal dimension. In Wan (1993) such a network is used for time series prediction, furthermore a backpropagation learning algorithm is developed.

In this thesis spikes trains are used instead of continuous signals. As a result, the signal processing with a FIR filter will not work at all. This is because the impulse response yields the temporal sequence of filter coefficients. Adding the filter coefficients up, results in the DC voltage gain of the filter. In case of an IF neuron without an absolute refractory period, the resulting activity of the target neuron is proportional to the activity of the spike emitting neuron, shifted in time by the maximum of connection delays. The proportionality factor is the sum over all weights, the DC voltage gain.

In case of a LIF, instead of an IF neuron, the membrane potential decays over time. This means that contemporaneously or within a short span of time incoming spikes are more effective than spikes distributed in time. Viewing the multiple connections again, there must be a pattern of delays which correlates with an specific output spike pattern of the pre-synaptic neuron. The easiest approach are two connections with the delays $d_{0,1,0}$ and $d_{0,1,1}$ and a difference of $\xi = |d_{0,1,1} - d_{0,1,0}|$ between these delays. If the pre-synaptic neuron fires twice with an ISI of ξ, the first spike taking the longer delay will reach the post-synaptic neuron contemporaneously with the second spike taking the path with the shorter delay. Thus the connection transmits spikes superiorly with ISIs equal to ξ, which is a band pass characteristic.

This effect can be intensified by inserting additional connections with $\xi = |d_{0,1,\mathrm{k}} - d_{0,1,\mathrm{k}+1}|$. Fig 3.4 displays the result of such an arrangement. In addition to the expected passband at $\nu_{\mathrm{a}} = 100$spikes/s, passbands arise at the harmonics $2\nu_{\mathrm{a}}$, $3\nu_{\mathrm{a}}$, ... etc. The efferent activity ν_{e} neuron has got a significantly higher amplitude at the harmonics, because the afferent stimulation is greater as well. If there is a further rise in the afferent activity, spikes reach the neuron with such short ISI that the membrane decay plays an increasingly minor role. More and more there is a linear interrelation between the input and output activity. Because equidistant delays are a very notional arrangement, these basic approaches shall be generalised.

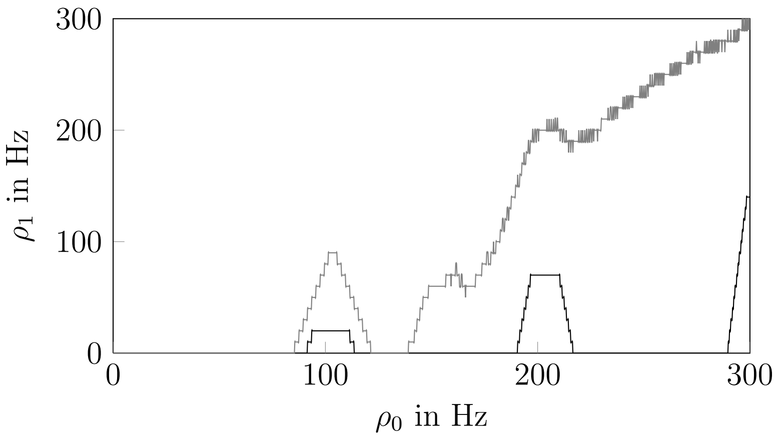


Figure 3.4: Moving average ρ_1 of the successor neurons spike train over the moving average of the predecessor's spike train ρ_0. Membrane time constant are $\tau_\mathrm{m} = 0.5$ms (black) and $\tau_\mathrm{m} = 3$ms (gray) and connections $d_{0,1,0} = 1$ms, $d_{0,1,1} = 11$ms, $d_{0,1,2} = 21$ms, $d_{0,1,3} = 31$ms, $d_{0,1,4} = 41$ms. Moving average parameters: integration window $\Delta t = 100$ms, discretization step width $\mathrm{d}t = 1$ms. Data recorded by increasing the predecessor's activity slowly with 200Hz/s.

3.2.2 A neuron as coincidence detector

Two LIF neurons are connected with K different delays $d_{0,1,\mathrm{k}}$ now. There exists a spike train pattern of the pre-synaptic neuron, which enables K spikes to reach the post-synaptic neuron simultaneously at time t_s. This pattern can be easily determined, the spike times are the reverse delays times $t_0^{(f)} = t_\mathrm{s} - d_{0,1,k}$. Note that one spike of neuron A leads to K spikes reaching neuron B, but more important than the number of spikes reaching a neuron, is their simultaneous arrival.

To investigate the phenomenon in more depth, the frequency response like in fig. 3.4 is not suitable. As an alternative approach a poisson impulse process[1] is used to determine the activity of neuron A, whereby the mean firing rate is $\bar{\nu} = 50$Hz. Its moving average is displayed in fig. 3.5 (top) and the successor's activity response is displayed below (fig. 3.5 (middle)). At the figure's bottom the correlation of the predecessor's random spike train with the optimal spike train is displayed. One can expect that the randomly

[1]A poisson process is a counting process, whereby the intervals between two counted events are exponential distributed. The poisson impulse process is the derivative of a poisson process. At each time the poisson process is incremented, the derivative is infinite and the derivative between two events is zero.

generated spike train resembles the optimal spike train from time to time. The correlation of both should be relatively high these times. Furthermore one can expect that the activity of neuron B increases at those times, too. This correlation can be observed at several times. However, it is possible that this effect is caused by punctual high activities of the predecessor. The optimal spike train would lead to a correlation $\max(\mathrm{Corr}[\mathrm{S}_{\mathrm{ref}}, \mathrm{S}_0]) = 6$, here the maximum is 3. One is able to see a tendency but it is not clear if results are caused by transient increase of the firing rate of the predecessor or by the correlation effect.

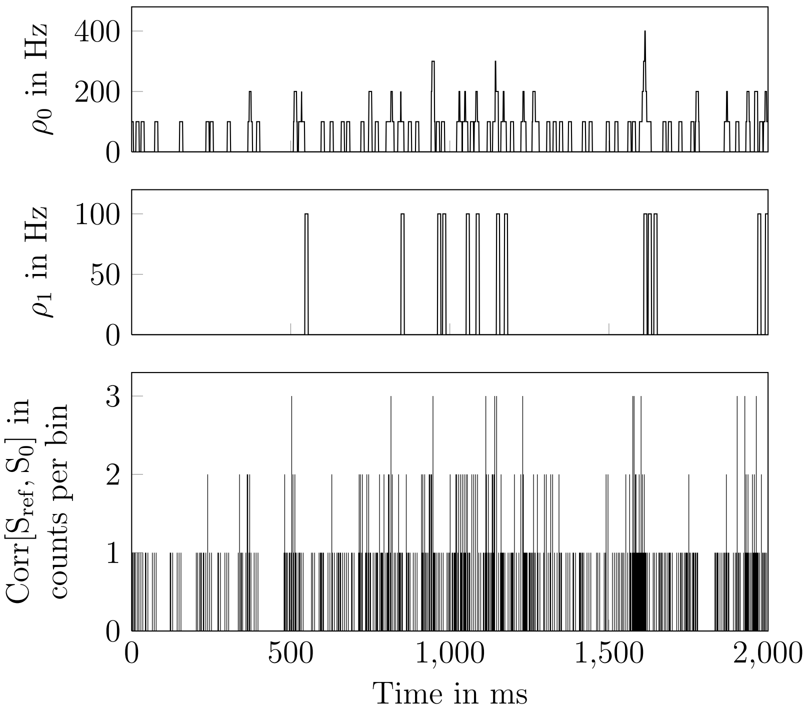


Figure 3.5: Moving average ρ_0 of the predecessor neurons spike train S_0 (top). N_0 fires randomly (poisson impulse process) with $\bar{\nu}_0 = 50\mathrm{Hz}$. Moving average ρ_1 of the successor neuron N_1 (middle), with membrane time constant $\tau_{\mathrm{m}} = 7\mathrm{ms}$. The neurons are connected via six synapses $d_{1,0,0} = 1\mathrm{ms}$, $d_{1,0,1} = 4\mathrm{ms}$, $d_{1,0,2} = 10\mathrm{ms}$, $d_{1,0,3} = 18\mathrm{ms}$, $d_{1,0,4} = 30\mathrm{ms}$, $d_{1,0,4} = 36\mathrm{ms}$, weights $w_{1,0,k} = 0.2$. The correlation (bottom) of the predecessor neurons spike train and the largest coincidence spike pattern $\mathrm{S}_{\mathrm{ref}}$, which is $t^{(0)} = 0\mathrm{ms} + \tau$, $t^{(1)} = 6\mathrm{ms}+\tau$, $t^{(2)} = 18\mathrm{ms}+\tau$, $t^{(3)} = 26\mathrm{ms}+\tau$, $t^{(4)} = 32\mathrm{ms}+\tau$, $t^{(0)} = 35\mathrm{ms}+\tau$, the time shifting variable τ is set to zero. Thus there are no negative delays and those bins are not displayed. Moving average parameters: integration window $\Delta t = 10\mathrm{ms}$, discretization step width $\mathrm{d}t = 1\mathrm{ms}$.

To achieve a correspondingly high correlation, eventually one has to wait for a long period time. It would be more advantageous to include the ideal spike train into the randomly generated signal. This is not a simple matter, too, because the deterministic sequence would disturb the stochastic process in any case. In fig. 3.6, for the sake of simplicity, a superposition of a poisson impulse process and the optimal spike train is used as the predecessor's spike train. The poisson impulse process, with a mean firing rate of $\bar{\nu}_0 = 100\text{Hz}$, is active within the first second. The optimal spike train is repeated every 0.5s. It is obviously that the successor neuron only reacts on the optimal spike train, the randomly arriving spikes are ignored.

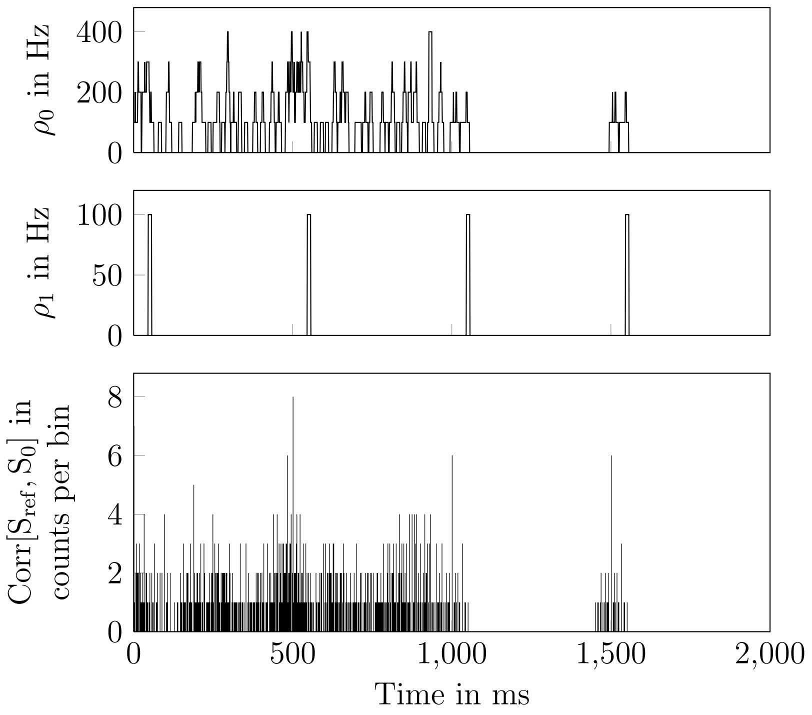


Figure 3.6: Similar to fig. 3.5, but neuron N_0 fires randomly (poisson impulse process) with $\bar{\nu}_0 = 100\text{Hz}$ for the first 1000ms. Furthermore N_0 fires the optimal spike train $t_0^{(0)} = 0\text{ms}+\tau$, $t_0^{(1)} = 10\text{ms}+\tau$, $t_0^{(2)} = 18\text{ms}+\tau$, $t_0^{(3)} = 32\text{ms}+\tau$, $t_0^{(4)} = 43\text{ms} + \tau$, $t_0^{(5)} = 50\text{ms} + \tau$ at $\tau = 0\text{ms},\ 500\text{ms},\ 1000\text{ms},\ 1500\text{ms}$. The membrane time constant of neuron N_1 is decreased to a level ($\tau_\text{m} = 2\text{ms}$) at which a response to the predecessor neuron is highly improbable if the input spike pattern is generated randomly. The two neurons are connected via six synapses again: $d_{1,0,0} = 1\text{ms}$, $d_{1,0,1} = 8\text{ms}$, $d_{1,0,2} = 19\text{ms}$, $d_{1,0,3} = 33\text{ms}$, $d_{1,0,4} = 41\text{ms}$, $d_{1,0,4} = 51\text{ms}$, weights $w_{1,0,k} = 0.17$. Moving average parameters: integration window $\Delta t = 10\text{ms}$, discretization step width $\text{d}t = 1\text{ms}$.

These results can be adapted to constellations with more than one predecessor neuron. Besides the firing times of the predecessor neurons, the delay times are important for the exact moment the spike arrives at the successor neuron. Assuming that all delays are in an equal range, the largest activity of the successor occurs if all predecessors are firing synchronously. If the delays are not in an equal range the largest activity of the successor occurs, if all predecessors are firing with an specific order (determined by the delays). Thus the successor neuron is working as a coincidence detector. To achieve a synchronous spike arrival time t_s at the successor, t_s must be counted back through the delays to get the predecessor's firing times $t_j^{(f)} = t_\mathrm{s} - d_{i,j,k}$. Unfortunately there are only a few analysis methods which are able to simultaneously analyse more than two spike trains, see section 2.1.1. Therefore this qualitative description is given here. Possibilities and potentials of coincidence detection are further described on the basis of an application in chapter 4.

3.3 Feed forward networks

Feed forward networks differ from recurrent networks in that connections do not form a directed cycle. In the connection matrix this implies that all entries below the main diagonal and on the main diagonal itself are zero. For practical application the absence of directed cycles means that the network behaviour is stable and does not form oscillations caused by feedbacks. Fig. 3.7 shows a schematic drawing of a two layer feed forward network.

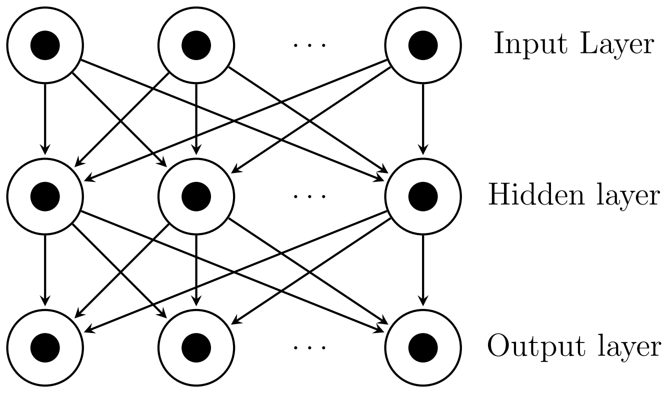


Figure 3.7: A two layer feed foward network.

Considering a two layer feed forward network with n_i input-, n_h hidden- and n_o output-neurons, each hidden neuron has got n_i predecessors and each neuron in the output layer n_h predecessors. Assuming that each neuron has got a similar activity on average, the number of predecessors can give an idea how to dimension the related weights. To consider the information of many predecessor neurons, weights should be small enough to allow a certain number of input spikes. A first indication for an IF neuron could be that on average half of the predecessors have fired once before the successor fires, then the weights would be on average $w \approx 0.5/n_\mathrm{i}$ between input and hidden layer and $w \approx 0.5/n_\mathrm{h}$ between hidden- and output layer. Involving the membrane decay of a LIF things are getting much more complicated. The period of time the input spikes arrive at the successor have to be taken into account. Instead of changing the weights, the membrane time constant τ_m can be adjusted to changing conditions. However, in the practical application in the following chapter 4 it has been very successful to choose a constant value of around $5 \ldots 20$ms for τ_m. But altogether, no general dimensioning basis can be specified here.

3.4 Recurrent networks

In recurrent networks connections between neurons may form directed cycles. This increases the potential number of neural connections immensely. On the one hand this can increase functionality and performance, but on the other hand oscillations in feedback occur. What leads to the problem of instability. It is known that the brain includes lot of directed cycles. A distinction in literature of biological recurrent neural networks is made in feedback which occurs within a single processing layer and feedback which occurs between multiple processing layers. In this thesis the focus is on feedback within a single processing layer. Biological examples for this are inter alia: short-term memory, winner-takes-all decision making, contrast enhancement and normalization (Grossberg, 2013).

3.4.1 Different types of feedback

In principle there are two possibilities to form directed cycles within a single layer. The first option is to loop the output of a neuron back to its input, what is called direct feedback. The second option is to connect the output of the neuron to the input of its neighbours, what is called lateral feedback. The two different types are shown in fig. 3.8. Note that feedback may not necessarily be excitatory, inhibition may also be possible.

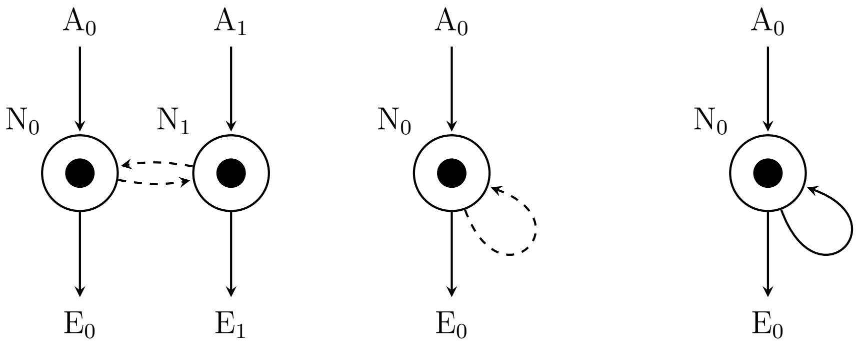


Figure 3.8: Lateral inhibition (left), with an input spike train A_0, N_0 transmits the stimulus to E_0 and simultaneously suppresses the neighbour path (A_1 to E_1) by inhibiting N_1. This applies in reverse as well. Recurrent inhibition (center), if the neuron is activated via A_0 and inhibits itself. Those negative feedback is a principle of self regulation and is able to enforce stability. Recurrent excitation (right), if the neuron is activated via A_0 and excites itself. Those positive feedbacks are able to grow the system activity like an avalanche, but are typically bounded by external conditions.

Direct feedback

Direct feedback can be divided into two types, positive (fig. 3.8 (right)) and negative feedback (fig. 3.8 (center)). The principle of feedbacks is used in many disciplines like electronic engineering, control theory, etc. Negative feedback allows finer tuning and increases stability, while positive feedback increases a system's agility.

Without any feedback, one would expect that the output firing rate $\bar{\nu}_{\text{out}}$ divided by the input firing rate $\bar{\nu}_{\text{in}}$ is approximately proportional to the input weight.

If feedback is negative, the membrane is reset or even negatively preloaded, depending on θ_{l} and the weight of the feedback $w_{i,i}$. This may suppress further spikes after one output spike occurred, in figure 3.9 this can be observed. With short delays the effect is very similar to the one of the refractory period. But the arbitrary delay offers the possibility to perform the inhibition at an arbitrary point in time. For inhibitory connections the cycle must always be stable.

If feedback is positive, the membrane is positively preloaded. This makes the neuron more sensitive to further incoming spikes, it is more likely to fire again. The arbitrary delay time offers the possibility to chose the point in time of self stimulation. The pre-loading is degraded by the membrane leaky current, thus this effect only remains for a short time. As long as there is only

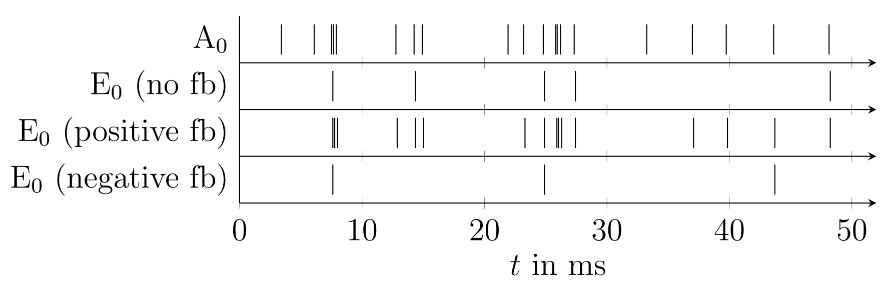


Figure 3.9: Direct feedback, cf. Fig. 3.8 (right). Input spike trains: $w = 0.3$, $\bar{\nu}_0 = 300\text{Hz}$ (poisson impulse processes). Neuron without direct feedback: $E_{0(\text{nofb})}$: $w = 0$, with positive feedback: $E_{0(\text{positivefb})}$: $w = 0.9$, $d = 0.1\text{ms}$, with negative feedback $E_{0(\text{positivefb})}$: $w = -0.9$, $d = 0.1\text{ms}$. Neuron parameters: $\vartheta_\text{l} = -1$, $\tau_\text{m} = 20\text{ms}$.

a single direct feedback and its weight is smaller than the upper membrane threshold, the cycle is stable. The process of self excitation is shown in 3.9. If the weight becomes equal or greater than the threshold, the cycle may become marginal stable, because each output spike will cause an input spike which again evokes an output spike. If there is more than one feedback path each output spike can evoke more than one input spikes. This could make the circuit instable.

Lateral feedback

Lateral inhibition can be found in many kinds of neural networks. For instance reciprocal inhibition antagonists in spinal motor structures or more complex applications like contrast enhancement in the visual system (Bear et al., 2008). Lateral excitations in neural networks have been found within the Olfactory Bulb or in the Escape Circuit of Crayfish for example. Corresponding publications can be found in neuroscientific journals, but are not part of and therefore will not be discussed further in this thesis.

As a minimal example, according to the scheme in figure 3.8 (left), two neurons inhibiting each other laterally shall be stimulated with randomly generated spike trains. The results of the simulation are shown in fig. 3.10. If a neuron is active, the activity of neighbour neurons is suppressed. In principle it is the same as the winner-takes-all network described in the following subsection.

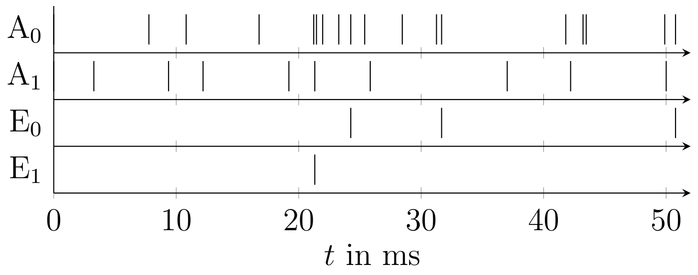


Figure 3.10: Lateral inhibition, cf. Fig. 3.8 (left). Input spike trains: $w = 0.3$, $\bar{\nu}_0 = 300\text{Hz}$ (poisson impulse processes). Lateral inhibiting connections: $w = 1$, $d = 0.1\text{ms}$. Neuron parameters: $\vartheta_\text{l} = -0.1$, $\tau_\text{m} = 15\text{ms}$.

3.4.2 A winner-takes-all network

Neurons of a winner-takes-all network are connected in a way that they are in competition. The competition offers the opportunity of decision-making. In a classical form the neuron with the highest activation wins this competition and forces the others to switch off. As the basis of decision various processes can be considered. Next to the highest activations, which refers to rate coding, it could be the time to the first spike or a designated input spike pattern.

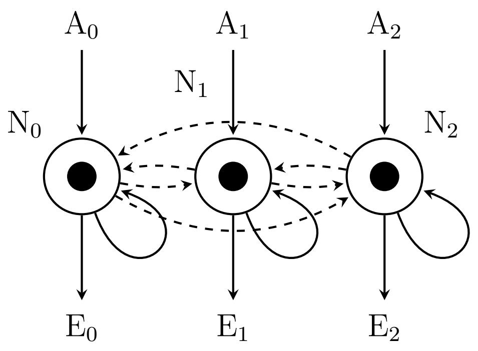


Figure 3.11: A winner-takes-all network of three neurons. If one of the neurons is activated by A_0, A_1 or A_2, the according neuron inhibits its neighbours (dashed arrows). In order to enhance and perpetuate the result of this computational principle, the neuron may exite itself (loops).

Fig. 3.11 shows a typical wiring of a winner-takes-all network with three competitive neurons. The suppression of the neighbour neurons takes place by the inhibiting lateral connections. To have a direct effect, the delay times of these connections should be chosen rather short, in the example shown in

Fig. 3.12 a delay of $d = 0.1$ms is taken. The weights of the lateral connections should be chosen rather high to bring an effect. Additionally the negative threshold of the neurons can be chosen very low, thus the inhibited neurons are negative preloaded and the inhibition effect persists longer. Reasonable values are $w = \vartheta_\mathrm{u} + |\vartheta_\mathrm{l}|$.

A further intensification of the competitive effect can be achieved by direct feedback, in order to let the neuron fire more easily again. These weights can be chosen rather high, too. If $w_{i,i} = \vartheta_\mathrm{u}$ the neuron activates itself continuously after it fired once. This marginal stable state makes it impossible or highly difficult to change the winner over time.

An exemplary demonstration is given in Fig. 3.12. The competing neurons are LIF neurons and the connection parameters are dimensioned in a way that the winner can change over time. At $t \approx 10$ms neuron N_1 is the first winner of the competition, but the following activation via A_1 is apparently not sufficiently high to keep the supremacy. At $t \approx 80$ms neuron N_0 is successful, but is not able to keep the supremacy, too. Thus the winner changes again in the further process.

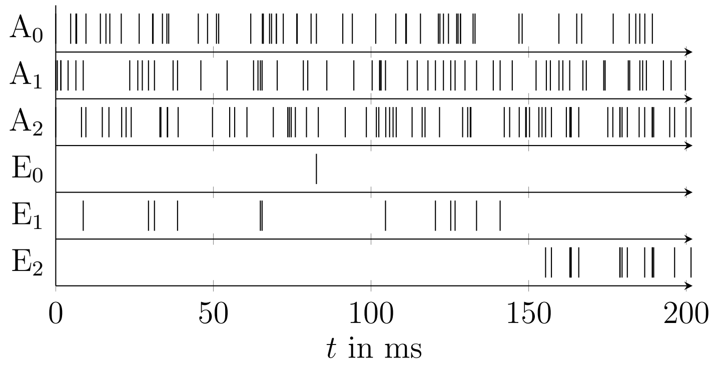


Figure 3.12: Winner-takes-all network, cf. Fig. 3.11. Input spike trains: $w = 0.2$, $\bar{\nu}_0 = 300$Hz (poisson impulse processes). Lateral inhibiting connections: $w = -1$, $d = 0.1$ms. Direct feedback connections: $w = 0.9$, $d = 0.1$ms. Neuron parameters: $\vartheta_\mathrm{l} = -0.5$, $\tau_\mathrm{m} = 20$ms, $\Delta_\mathrm{abs} = 0$ms.

3.4.3 Memory

The way how information is stored inside the brain is largely unknown. It is assumed that memory takes place in most parts of the brain, whereby it is expected that specific brain regions are responsible for different types of learning, which are sensory memory, short-term memory and long-term memory. Besides modification of synaptic weights (which can be understood as learning and memory as well), the way neurons are connected with one another facilitates the storage of information as well.

Here a very simple and artificial example of neural wiring, which provides an opportunity to store information, is given. Fig. 3.13 shows an aggregate of five neurons, which are connected in a ring structure. The weights are adequate to evoke a spike in the post-synaptic neuron by a single pre-synaptic neuron. Information inserted into the neuron circle via input A_0 is circling around, until it is erased with the inhibitory input signal A_1. Data that is saved in this way can be retrieved periodically at each neurons output.

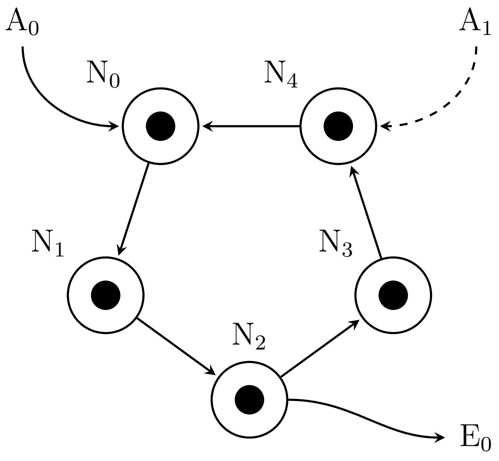


Figure 3.13: Aggregate of five neurons in order to store activation patterns. An afferent activation pattern can be inserted into the neuron circle by input A_0. The stored pattern appears cyclically at E_0, efferences of the other neurons are suitable as well. Inhibition by the afference A_1 results in a clearance of the circling activation pattern. Parameters: $w_{i,j} = \vartheta_{\mathrm{u}} = 1$

In Fig. 3.14 a random spike pattern is loaded into the cycle. The length of the input spike pattern is limited by the sum of the used delays inside the circle, otherwise the end overlaps the begin and the spike train is muddled in consequence. One can clearly see how this pattern appears delayed ($d_{i,j} = 5$ms) at one neuron after the other. On the one hand this example demonstrates the effect of memory, on the other hand the example confirms the correct operation of the software for recurrent networks.

At $t = 100$ms an inhibitory spike burst appears at input A_1 and deletes the stored information. When dimensioning the inhibition process, one has to take care of the frequency and length of the spike burst, the weight of the according connection and the negative threshold of neuron N_4. In addition to the connection weight, the frequency of the spike burst is significant for the

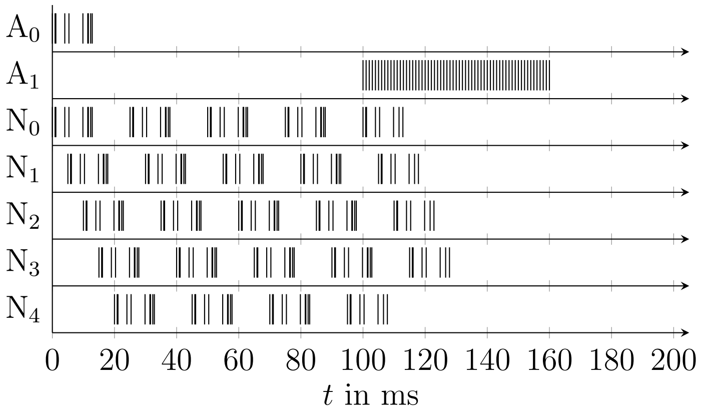


Figure 3.14: Storage of a spike train by an aggregate of five neuron, see Fig. 3.13. An spike pattern is inserted by A_0 into the neuron cycle at $t = 0$ms. At $t = 100$ms the circling spike spattern is erased by an inhibition caused by A_1. Parameters are $w_{i,j} = 1$, $d_{i,j} = 5$ms and $\vartheta_\mathrm{l} = -1$, $\vartheta_\mathrm{u} = 1$, $\Delta_\mathrm{abs} = 0.1\mu$s, $\tau_\mathrm{m} = \infty$

strength of the inhibition effect, see temporal summation in Eq. 2.18. The length of the inhibiting spike burst must be greater or equal to the maximal length of the stored spike pattern or to the sum of the used delays inside the circle respectively. This ensures that no information remains. Furthermore it could be advantageously to enlarge the duration over several passes, thus any remaining spikes are deleted, too. Here again a negative preloading and the related ruggedness against excitatory peaks can be achieved through a larger negative threshold.

3.4.4 A fully connected network – influence of parameters and computing time

As basis of this investigations a fully connected network with 25 neurons is generated. The weights ($w_{i,j} = \pm 0.1 \ldots 0.2$) of the connections are uniformly distributed, whereby 10% are inhibitory and have a negative sign. The delays ($d_{i,j} = 1 \ldots 5$ms) are also uniformly distributed. In order to allow a little negative preload, the neurons' lower thresholds are $\vartheta_\mathrm{l} = -0.5$. Furthermore a membrane $\tau_\mathrm{m} = 15$ms is given for all neurons. To initiate an activity of the network, the first ten neurons fire randomly (poisson impulse process) with $\bar{\nu} = 150$Hz for the first 25ms. The simulation time is limited to $t = 500$ms.

Totally different behaviour of one and the same network but different initiation spike trains can be observed. In fig. 3.15 an example is given in which the activity fades away very quickly. After only $30 \ldots 40$ms the network is back again in a state of calmness. Figure 3.16 shows a round with some other initiation spike trains. The network activity increases within the first ≈ 100ms, after that the resulting pattern remains. No alteration can be observed after 500ms (not displayed). This indicates that the network is getting into a marginal stable state. After several runs no result could be observed, in which the network's activity runs into an unstable state. Although such a state might be possible due to the huge number of feedbacks.

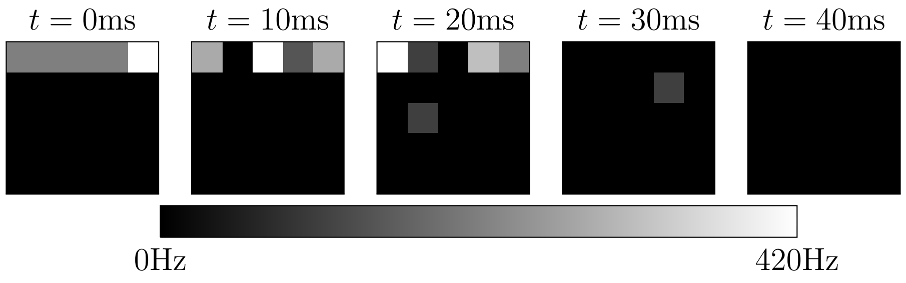


Figure 3.15: Activity heatmap of a fully connected network with 25 neurons, black corresponds to $\nu = 0$ and white to $\nu = 420$Hz. Initiation spike trains of neuron $0 \ldots 9$ are randomly generated (poisson impulse process), $\bar{\nu} = 150$Hz. Neuron parameters: $\vartheta_\text{l} = -0.5$, $\tau_\text{m} = 15$ms, $\Delta_\text{abs} = 0$. Connection parameters: $w_{i,j} = \pm 0.1 \ldots 0.2$, $d_{i,j} = 1 \ldots 5$ms, 10% of the connections are inhibiting. Moving average parameters: integration window $\Delta t = 100$ms, discretization step width $\mathrm{d}\,t = 1$ms. Computing time: 0.03s

Computing time

Here the computing time is measured by calling the clock_t clock(**void**) function of the time.h before and after the simulation and calculating the difference. The result is returned in clock ticks and therefore is divided by the constant CLOCKS_PER_SEC in order to convert it into the time format. This is a very simple measurement method but sufficiently precise in this application. As hardware a Pentium(R) Dual-Core CPU E5200 @ 2.50GHz 2 and 3.2GiB ram was used with an Ubuntu 12.04 (precise) (32-Bit) (Kernel Linux 3.2.0-57-generic-pae) operating system.

A comparison of the two simulations in fig. 3.15 (0.03s) and 3.16 (42.74s) shows a significant difference between computing times. This difference cannot be explained by the network's neuron and connection parameters because they are unaltered. The reason for this lies in the different neural activities. One has to consider that not only the processing of the spikes does take time. The most time consuming element is to insert the output spikes into the target neuron's spike container. Inserting F_i spikes into a container with size F_c goes logarithmic with $F_\mathrm{i} \log(F_\mathrm{i} + F_\mathrm{c})$.

A further effect on the computing time occurs if there are feedbacks with short circling time in conjunction with a high neural activity. Then the neurons' simulation function has to be called very often, cf. sec. 2.3.

Larger networks are getting extensive very quick. The number of connections increases the number of spikes as well. A fully connected network with one hundred neurons cannot be sufficiently studied with this software and the used PC because it exceeds the 3.2GiB memory quickly.

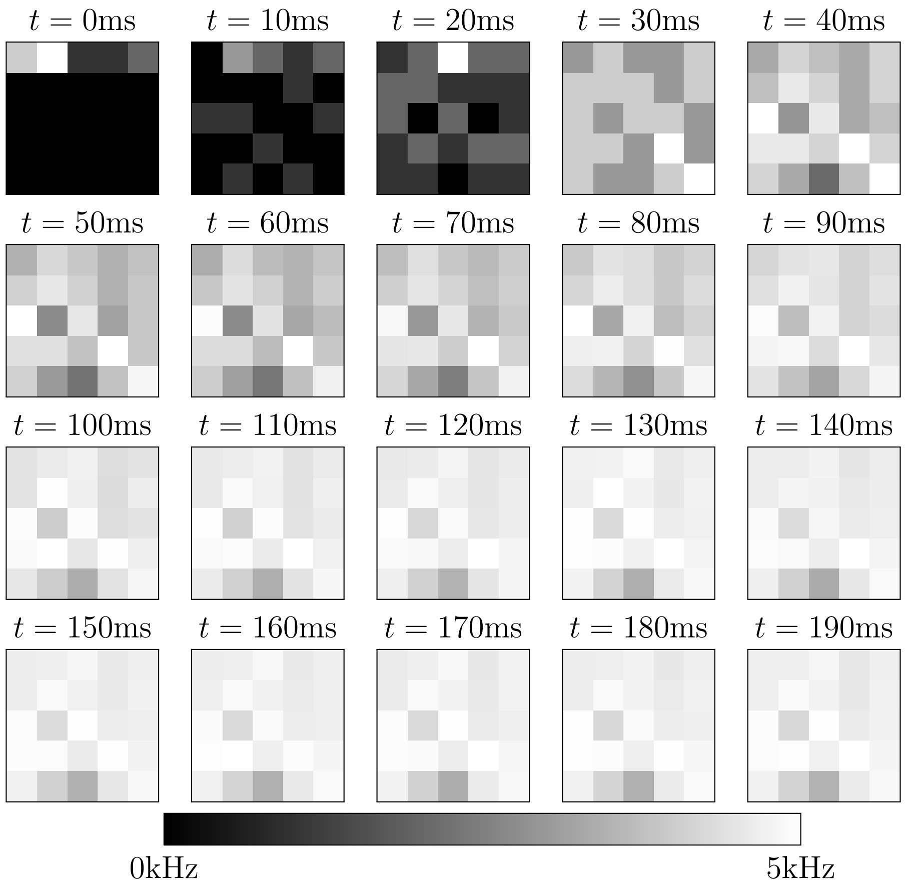


Figure 3.16: Activity heatmap of a fully connected network with 25 neurons, black corresponds to $\nu = 0$ and white to $\nu = 5$kHz. Identical to network in fig. 3.16, but different initiation spike train. Initiation spike trains of neuron $0 \ldots 9$ are randomly generated (poisson impulse process), $\bar{\nu} = 150$Hz. Neuron parameters: $\vartheta_\mathrm{l} = -0.5$, $\tau_\mathrm{m} = 15$ms, $\Delta_\mathrm{abs} = 0$. Connection parameters: $w_{i,j} = \pm 0.1 \ldots 0.2$, $d_{i,j} = 1 \ldots 5$ms, 10% of the connections are inhibiting. Moving average parameters: integration window $\Delta t = 100$ms, discretization step width $\mathrm{d}\,t = 1$ms. Computing time: 42.74s

3.5 Synaptic plasticity - The ability to learn

The ability of neural systems to change their behaviour over time, with the aim to improve capability and performance, is generically referred to as learning. The underlying alteration process of the neural system is called neuro-plasticity or brain-plasticity. A distinction is made between synaptic and non-synaptic plasticity. Synaptic plasticity is described as the ability of synapses to strengthen or weaken over time, whereby the increase of strength of a synapse is called Long-Term-Potentiation (LTP) and the decrease is called Long-Term-Depression (LTD). This process is thought to be a major part of learning and memory. Non-synaptic plasticity is less studied. It refers to the ability of changes in the characteristics of the remaining cellular components like soma, axon and dendrites.

A further distinction is made between the time frames within the alteration process takes place:

- Short term plasticity
  Alteration remains for a sub millisecond period.

- Long term plasticity
  Alteration remains for minutes, hours, days or even longer.

The processes behind are rather complex and due to time restrictions it would not be suitable to be concerned with all of them in detail. In this thesis solely synaptic weight changes in terms of long term plasticity are given priority. An excellent starting point for this is given by neuro-psychologist D. O. Hebb in his book "The Organization of Behaviour" from 1949. He postulated the following:
"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 change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."(Hebb, 1949)
This sentence is often vaguely summarised by "what fires together, wires together". It underlines Hebb's idea that learning is based on correlation. In the following two mathematically interpretations of this rule are given. The first one is a rate based and the second one a spike time based learning concept.

3.5.1 Rate based Hebbian learning

The general approach for the synaptic plasticity is an arbitrary function of the activities ν_i and ν_j of a pre-synaptic neuron j and a post-synaptic neuron

i, connected by a synapse with the weight w_{ij}. For a rate based learning the activities can be assumed as firing rates.

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = F(w_{ij}; \nu_i, \nu_j) \tag{3.3}$$

In Gerstner et al. (2008) F is expanded in a Taylor series about $\nu_i = \nu_j = 0$:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = {} & c_0(w_{ij}) + c_1^{\mathrm{post}}(w_{ij})\nu_i + c_1^{\mathrm{pre}}(w_{ij})\nu_j \\ & + c_2^{\mathrm{pre}}(w_{ij})\nu_j^2 + c_2^{\mathrm{post}}(w_{ij})\nu_i^2 + c_2^{\mathrm{corr}}(w_{ij})\nu_i\nu_j + \mathcal{O}(\nu^3) \end{aligned} \tag{3.4}$$

The simplest form is to set all terms but the correlation term to zero, which is *Hebb's hypothesis*:

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = c_2^{\mathrm{corr}} \nu_i \nu_j \tag{3.5}$$

This equation is sometimes called activity product rule, whereby the constant factor c_2^{corr} is called rate of learning. A disadvantage of Hebb's native rule is the missing of negative weight changes, thus the weights are driven into saturation over time. Some more complex learning rules can be deduced from equation 3.4 and remove the problem. These are for example the covariance rule and the Bienenstock-Cooper-Munroe rule (Haykin, 1994).

3.5.2 Spike-Time-Dependent-Plasticity (STDP)

By assuming that not the firing rate but the exact spike timing is the basis of information, another learning algorithm is required. Unlike in 3.3, in STDP weight changes are given by a function of pre- and post-synaptic spikes.

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = f(w_{ij}; t_i^{(f)}, t_j^{(f)}) \tag{3.6}$$

A general model with pre- and post-synaptic spike trains $S_j(t) = \sum_f \delta(t - t_j^{(f)})$ and $S_i(t) = \sum_f \delta(t - t_i^{(f)})$ can be denoted:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t} w_{ij} = a_0 & + S_{\mathrm{j}}(t)\left[a_1^{\mathrm{pre}} + \int_0^\infty a_2^{\mathrm{pre,post}}(s) S_i(t-s)\,\mathrm{d}s\right] \\ & + S_{\mathrm{i}}(t)\left[a_1^{\mathrm{post}} + \int_0^\infty a_2^{\mathrm{post,pre}}(s) S_j(t-s)\,\mathrm{d}s\right] \end{aligned} \tag{3.7}$$

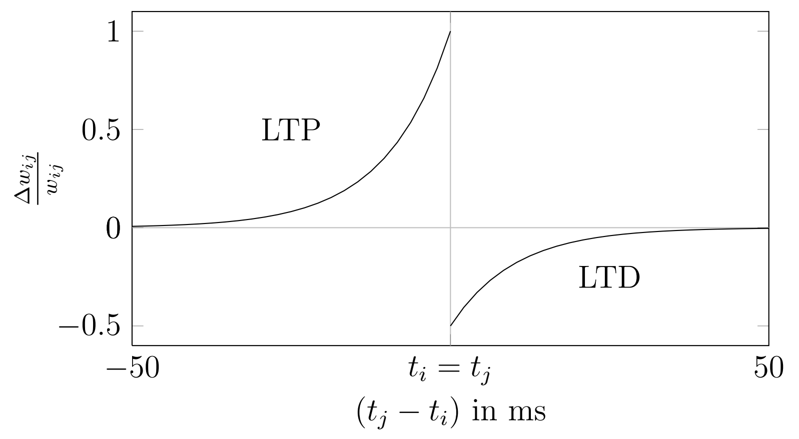


Figure 3.17: STDP with an exponential learning window ($A_+ = 1$, $A_- = 0.5$, $\tau_+ = \tau_- = 10$ms). Synaptic efficiency alterates if pre-synaptic spike at $t_j^{(f)}$ and post-synaptic firing at $t_i^{(f)}$ are temporal close. Synaptic efficiency is increased (LTP) if the pre-synaptic spike precedes the post-synaptic spike and is decreased (LTD) for reversed timing.

Parameters a_0, a_1^{pre}, $a_2^{\text{pre,post}}$ and kernels a_1^{post} have got the following meanings:

a_0	Spike independent term (non–Hebbian)
a_1^{pre}	Pre-synaptic plasticity (non–Hebbian), refers to presynaptic spikes in ignorance of post-synaptic spikes
$a_2^{\text{pre,post}}$	Correlation based plasticity (Hebbian), refers to post-synaptic spikes occurring before pre-synaptic spikes
a_1^{post}	Post-synaptic plasticity (non–Hebbian), refers to post-synaptic spikes in ignorance of pre-synaptic spikes
$a_2^{\text{post,pre}}$	Correlation based plasticity (Hebbian), refers to post-synaptic spikes occurring after pre-synaptic spikes

A common choice is to set the non-Hebbian terms to zero and to take an exponential function for the two Hebbian terms $a_2^{\text{pre,post}}$ and $a_2^{\text{post,pre}}$. The resulting "learning window" $W(s)$ is

$$W(s) = \begin{cases} A_+ \exp(-s/\tau_+), & \text{for } s \geq 0 \\ -A_- \exp(s/\tau_-), & \text{for } s < 0 \ . \end{cases} \tag{3.8}$$

The time constants are in the range of $\tau_+ = 10$ms and $\tau_- = 10$ms, which are slightly smaller but correspond to the time constants of the neuron membranes τ_{m}. Note that this learning algorithm finds its utilization for excitatory synapses only. STDP processes in inhibitory synapses are less well studied because inhibitory synapses are rare. Consequently no learning algorithm for inhibitory synapses is established, these connections remain unaltered.

The total weight change can be denoted by the sum over all weight changes caused by pairs of pre- and post-synaptic spikes.

$$\Delta w_j = \sum_{m=1}^{N} \sum_{n=1}^{N} W\left(t_i^{(n)} - t_j^{(m)}\right) \tag{3.9}$$

As long as A_+ and A_- are not a functions of w_{ij} weights cannot be bounded. If a weight of a synapse becomes greater than the upper threshold of the neuron ϑ_u, each spike running over this synapse forces the post-synaptic neuron to spike. Thus the weight of this synapse enlarges more and more. Naturally unbounded weights are not truly biologically realistic, too.

In order to keep the weights within an interval $w^\mathrm{min} < w_j < w^\mathrm{max}$, different weight dependent amplitudes $A_+(w_j)$, $A_-(w_j)$ are conceivable. To simplify, the lower bound is set to zero, w^min. One possibility is a simple linear function, called soft bounds or multiplicative weight dependence:

$$A_+(w_j) = \eta_+(w^\mathrm{max} - w_j) \tag{3.10}$$

$$A_-(w_j) = \eta_- w_j \tag{3.11}$$

Besides linear soft bounds more complex function traces are possible. Another possibility is called hard bounds. The weights are rigidly restricted by the bounds. Mathematically this can be expressed with the Heaviside step function Θ:

$$A_+(w_j) = \eta_+\Theta(w^\mathrm{max} - w_j) \tag{3.12}$$

$$A_-(w_j) = \eta_-\Theta(-w_j) \tag{3.13}$$

Further information on the subject of STDP can be found in Sjöström and Gerstner (2010) and Gerstner et al. (2008).

A further way to keep the weights stable was tried but achieved no satisfactory results. The idea was to keep the sum-of-squares of all weights leading towards a neuron constant. Of course the weights are stable with this procedure, but a problem is that connections which should actually be increased by learning can be decreased by the following compensation. To keep the sum-of-squares of all connections leading to the neurons' successors constant was not tested. May be this is less critical due to the reduced number of connections per neuron, but in general the described problem remains.

Influences of STDP

STDP can be regarded as an enhanced interpretation of the Hebbian learning rule for SNNs (section 3.5). As described in chapter 2.1, information in SNNs could be coded within the firing rates or the exact spike times. The STDP has got a direct influence on the firing rate of a neuron and its firing time as well.

The influence on the firing rate is obvious. Increasing weights enable that less input spikes are needed to evoke an output spike, resulting in higher firing rates. Decreasing weights effect opposite. Since the exponential STDP learning window used here evokes positive as well as negative weight changes, the η_+/η_- ratio has got a main impact on the alteration of the mean firing rate of the neurons. In conclusion the STDP leads to a greater sensitivity (and a resulting greater firing rate) to inputs from the connection which evoke an output spike before.

The influence on the spike times is a bit less obvious than the influence on the firing rate. One significant effect can be observed if one neuron is connected to a group of predecessors which are firing one after another (Sjöström and Gerstner, 2010). The target neuron shall spikes only once, after the fth input spike. Through the STDP learning process the connections which have been activated before the successors spike event are strengthened. After one or more repetitions the successor neuron may already fire after the $(f-1)$th input spike. These considerations demonstrate that STDP has a direct effect on the firing time.

Supervised learning and backpropagation

For a supervised learning method it is necessary to define an error in the output-layer. In the following chapter each output neuron represents one decision. The neuron which should win, whether it wins the competition or not, is reinforced by the STDP learning algorithm. This done by applying the STDP algorithm only to connections leading to the neuron which should win.

In networks with one or more hidden layers, the weights between input- and hidden-layer cannot be taught without defining a target constellation for the hidden layer. There is no possibility to propagate the failure from the output back to hidden neurons.

Supervised learning is used in the following chapter. The STDP algorithm from section this section serves as a basis. Because of the lack of a possibility to propagate the error back to a hidden layer, feedforward networks with one input and one output layer will take into account exclusively.

Some tests without supervising, so called clustering, has been performed without success. Also it appears possible on principle.

Chapter 4

Using the network for image recognition

Since the fundamentals have been pointed out in the previous two chapters, this chapter deals with a possible application of a SNN. The goal is the recognition of handwritten digits in an 8-bit gray value image. In a first instance the used procedures of neural encoding and decoding are described (sec. 4.1). After that the recognition of some less fractured but similar shapes is realised (sec. 4.2). This deals with intersections in the presented patterns in order to study the necessity of inhibiting connections. Conclusively the digit recognition is described (sec. 4.3).

4.1 Encoding and decoding

Before the recognition begins, a reasonable approach for neural encoding and decoding must be defined. It seems to be relatively clear that each pixel of the input image should be represented by one input neuron. The more interesting question is how to transform the gray values into spike patterns. As already mentioned in section 2.1.2, this is not an uncontroversial task. To deal with the dualism of rate and temporal coding, two procedures are used.

Encoding

The first procedure refers to rate coding. Each input neuron should fire with a frequency proportional to the gray value.

$$\nu_{x+y} = p(x,y)\frac{\nu_{\max} - \nu_{\min}}{p_{\max}} + \nu_{\min} \tag{4.1}$$

Whereby $\nu_{\min}$ is the lowest and $\nu_{\max}$ is the highest possible frequency, $p_{\max}$ is the maximal gray value. To avoid that all neurons fire the first spike at the same time, a normal distributed ($\mu = 20\text{ms}$, $\sigma = 5\text{ms}$) phase is added. In the software ISIs are used instead of firing rates.

The second procedure refers to the time-to-first-spike coding approach (see sec. 2.1.2). Each input neuron fires once. For this, the gray value of a pixel $p(x, y)$ is transformed linearly into a spike time.

$$t^{(0)}_{x+y} = p(x, y)\frac{t_{\max} - t_{\min}}{p_{\max}} + t_{\min} \tag{4.2}$$

Whereby $t_{\min}$ is the earliest and $t_{\max}$ is the latest possible time of spike occurrence.

The here given parameters are the same for all following investigations:

- Time $t_{\min} = 10\text{ms}$, $t_{\max} = 90\text{ms}$
- Firing rate $\nu_{\min} = 1/(3\text{ms})$, $\nu_{\max} = 1/(35\text{ms})$
- Gray value $p_{\max} = 255$

Decoding

Each decidable input pattern is represented by one neuron in the output layer. A winner in this layer indicates which pattern is represented at the network's input. For the decoding two possibilities can be used to determine the winner neuron in the output layer. One referring to rate coding and the other to temporal coding. The rate code winner could be determined by counting the output spikes of each neuron, the one with the largest number wins. The temporal decoding winner is determined by getting the time-to-first-spike, the neuron which is firing first wins. The competition is done by evaluation after the end of the simulation. Decidedly networks like the winner-takes-all network from section 3.4.2 are not used because of the time consuming simulation. If two neurons fire at exact the same time or fire the same number of spikes respectively the result is considered as false.

To enable the decoding with spike rates it should be the goal to increase the connection weights which belonging to the intersection set of all the digits of one group. The remaining connections are not needed. This can be achieved rudimentary with time-to-first-spike encoding, see sec. 4.3.2. Especially if all weights are initialized with very low values and the negative learning rate is set to zero, the requested receptive fields can be achieved. But in several tests the winner determination with spike rate codes failed. Regarding for example the results in fig. 4.10, the number of output spikes of the three neurons is very similar ($n_{784} = 9$, $n_{785} = 8$ and $n_{786} = 9$), the winner is not unambiguously determined because $n_{784} = n_{786}$. A reason for this is the numerousness of equal gray values in the represented images. Even in combination with strongly varying weights it results in a very similar mean input spike density. This problem remains, even after many training epochs and adapted weights. Here the STDP algorithm has only effects on the firing time of an output neuron. In case if it fires earlier or later. An alteration of the number of spikes is not possible because the number of connections are very high and grouped weight changes by STDP do not achieve the desired effect. In principle it should be possible to decode the information by the number of spikes but the STDP algorithm does not seem to be adequate to train these networks in that way. Therefore all winners are determined with time-to-first-spike decoding.

4.2 Recognition of similar shapes

The goal in this section is to enable the network to recognize the four shapes listed in fig. 4.1. These four shapes consist solely of the intersection of each other and therefore have a high degree of overlap, what makes them not easily distinguishable. The difficult task of separation may require negative weights, in order to make the small differences in the patterns more meaningful.

Each of the 28^2 pixel is represented by one input neuron and each decision is represented by one of the four output neurons. A connection exists between each input and output neuron. The weights are initialized uniformly distributed within the interval $[0.01, 0.011]$ and the delays within the interval [1ms, 9ms]. The neurons of the output layer have a lower threshold $\vartheta_{\mathrm{u}} = -0.1$ and a membrane time constant $\tau_{\mathrm{m}} = 15\mathrm{ms}$. The refractory period is without influence, because the winner is determined by the time-to-first-spike method (see sec. 4.1). Only the first spike is used in the learning process.

4.2.1 Pixel gray values encoded as spike times

The input neurons are only firing once in accordance to their gray value. The learning algorithm is applied to all connections relating to the neuron which should be the winner in each epoch. The parameters are $\eta_+ = 0.003$, $\tau_+ = 10\mathrm{ms}$, $\eta_- = 0.0018$, $\tau_- = 25\mathrm{ms}$ and $w^{\max} = 0.1$. All parameters, inclusively the neuron and connection parameters, have been determined through trial and error.

To simplify the task, only shape one and two should be recognized in a first step. The results vary widely. In only one half of all cases the decision-

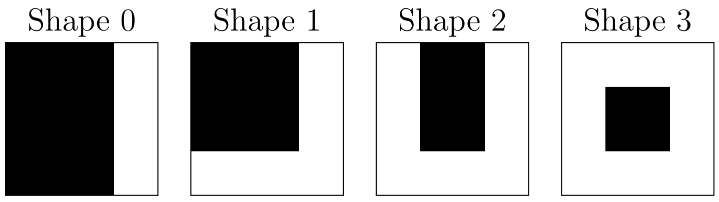


Figure 4.1: Four similar shapes to test the network.

making is correct within a fix number of epochs. Therefore another abort criterion is chosen here. Within each epoch it is tested if the two shapes can be distinguished from one another. If the decision-making is correct the learning process is aborted. If there is no correct decision after 200 epochs, the learning process is stopped, too. To examine the mean number of epochs which are necessary to make the network learn to recognise the two shapes, the program is executed five thousand times. The results show that the learning process was only successful in approximately 74.2% of the cases. The network needed on average 18.713 epochs to learn the two shapes. Some of the networks were accidentally initialized correctly.

To study the influence of negative weights, now 15% of the connections are inhibiting and get a negative sign respectively. The result of five thousand runs shows approximately 76.6% successful learning processes. On average 19.612 epochs were needed to make a network recognize the first two shapes correctly. If the same network is trained with all four shapes, the results are very bad. None of 5000 training runs have been successful, with or without negative weights.

Based on this examples, it cannot clearly be assessed that the used network is in principle able to solve the problem. Inhibitory connections appear not to be necessary for this task. Eventually a problem occurs due to the huge number of input neurons and the comparably small number of decisions. The learning algorithm is not suitable to adapt such a huge number of connections in a correct way with such a low rate of information. An immensely more extensive database provides the MNIST collection of hand written digits, whose recognition is topic of the next chapter. Furthermore there may be very large potentials in the optimisation of the parameters. Larger test series with a more powerful computer may be expedient.

4.2.2 Pixel gray values encoded as spike rates

The same network, which is used before, is taken here, but the gray values of the image pixels are encoded as spike rates. The learning process is aborted if there is no correct decision after 300 epochs. The results are significantly better, all 5000 learning processes succeeded. 92.914 epochs were needed on average to get the correct result. With 20% inhibiting connections the results are not better. 4924 out of 5000 successful learning processes (98.48%) could be observed. 138.011 epochs were needed on average to get a correct result. Fig. 4.2 displays the steep learning of one training run with inhibiting connections. The reason for the better results cannot be determined at this point.

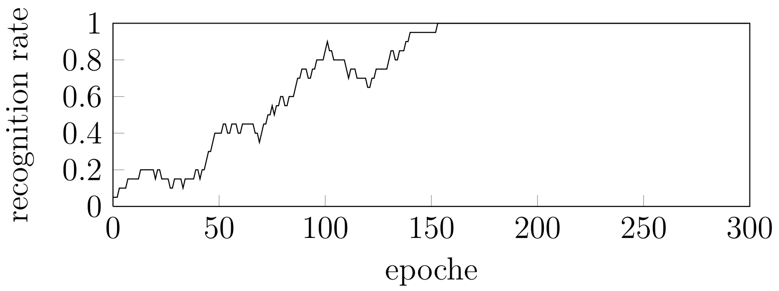


Figure 4.2: A network (without inhibiting connections) is learning the four shapes of fig. 4.1, pixel gray values are encoded as spike rates.

4.3 Recognition of hand written digits of the MNIST database

In the first subsection of this section, a network is used to recognize the handwritten digits of the MNIST database encoded as firing rates (subsec. 4.3.1). Within the second subsection 4.3.2, a comparison to pixels' gray values encoded as spike times is made.

The MNIST database is available in the internet[1], it has a training set of $60,000$ examples and a test set of $10,000$ examples. In this thesis only the digits 0, 1 and 2 are used. Furthermore a subset of the selected digits is used, which is always taken from the beginning of the MNIST training and test sets.

4.3.1 Pixel gray values encoded as spike rates

In figure 4.3 a heatmap of the input layer's activity at several points of time is displayed. Each pixel of the heatmap is representing one neuron. The gray values correspond to the current activity, which is calculated by

[1]`http://yann.lecun.com/exdb/mnist/`

a moving average function. For further information see section B.7. One sees that all white pixels which belong to the written digit are firing with a constant rate of $\nu = 1/(3\text{ms})$. The small firing rate of the background pixels induces a beat frequency in the pictures, because the integration window of the moving average function cannot be chosen arbitrarily long. This causes varying spike rates in the figure, in fact the background fires with a constant rate of $\nu = 1/(35\text{ms})$.

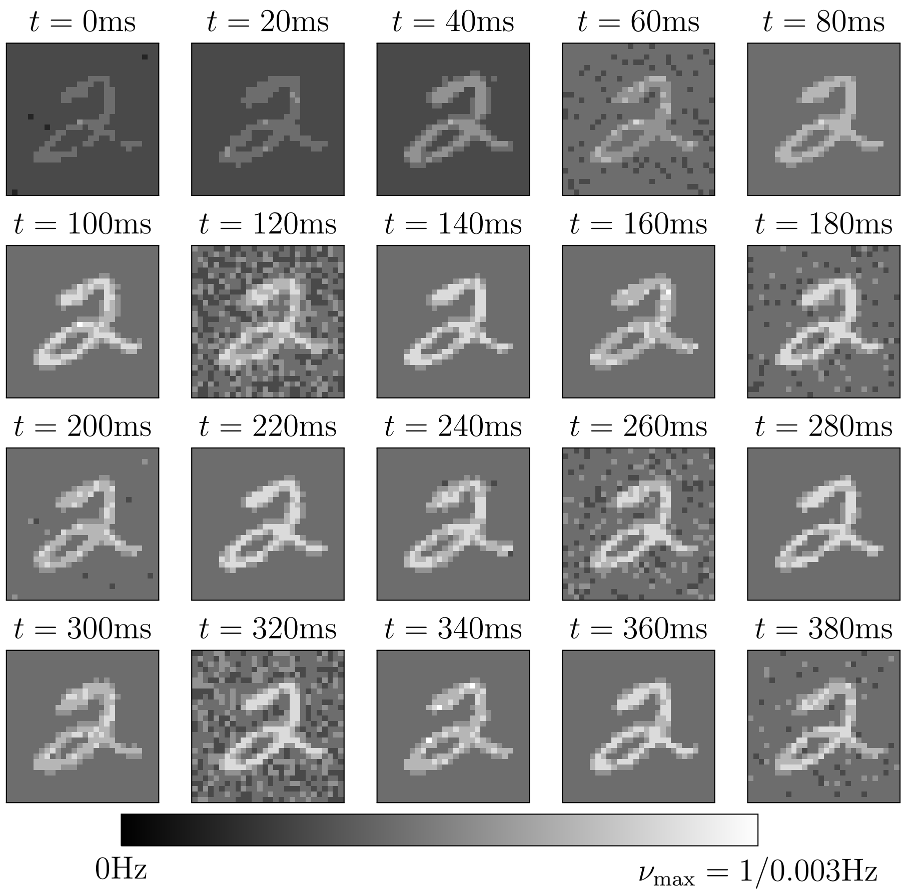


Figure 4.3: Exemplary activity heatmap of the input layer with 28^2 neurons and a handwritten "two" of the MNIST database. Spikes are generated according to equation 4.1, $\nu_{\min} = 1/(35\text{ms})$, $\nu_{\max} = 1/(3\text{ms})$ $p_{\max} = 255$. Moving average parameters: integration window $\Delta t = 200$ms, discretization step width $\mathrm{d}\,t = 1$ms.

The huge number of spikes generated in the input layer arrives at each neuron of the output layer. A visualisation of such a large number of spikes is not possible. Therefore only the output spikes of the output layer are displayed in figure 4.4. One can see that the number of output spikes is identical for each neuron in this example. The times to first spike differ slightly.

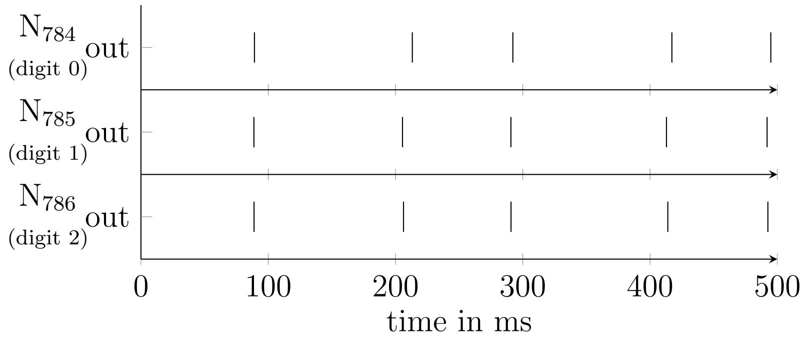


Figure 4.4: Exemplary spike trains of the three output neurons N_{784}, N_{785} and N_{786}. The number of input spikes is too large (approximately 12,500) to realize a visual separation. Therefore input spikes are not displayed.

In order to study rate coding and avoid the influence of coincidences simple IF neurons are used in this subsection. Neuron parameters are $\vartheta_{\mathrm{u}} = 1$, $\vartheta_{\mathrm{l}} = -0.1$ and $\Delta_{\mathrm{abs}} = 1\mathrm{ms}$. The connections are initialized uniformly and distributed with $w_{i,j} \in [1 \cdot 10^{-3}, 11 \cdot 10^{-3}]$ and $d_{i,j} \in [1\mathrm{ms}, 2\mathrm{ms}]$.

The network is trained over 3000 epochs with the STDP algorithm (learning rates: $\eta_+ = 2 \cdot 10^{-3}$, $\eta_- = 2.5 \cdot \eta_+ = 5 \cdot 10^{-3}$, time constants: $\tau_+ = \tau_- = 5\mathrm{ms}$, weight maximum: $w^{\max} = 0.05$). Only the first spike of an output neurons spike train is considered. The training set consists of 300 examples (100 times digit 0, 100 times digit 1, 100 times digit 2) and the test set consists of 1500 examples (500 times digit 0, 500 times digit 1, 500 times digit 2). The decoding in the output layer is done by time-to-first-spike.

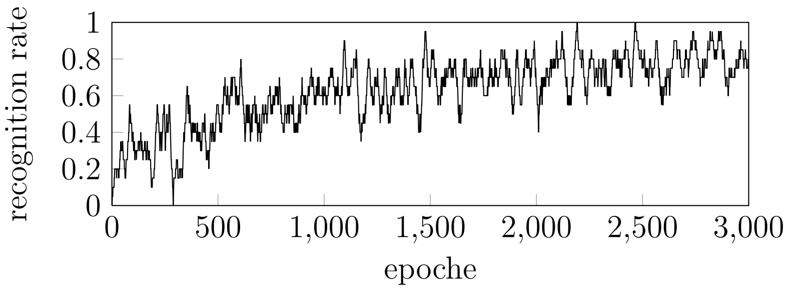


Figure 4.5: Learning process of a training set from the MNIST database, pixel gray values are encoded as spike rates.

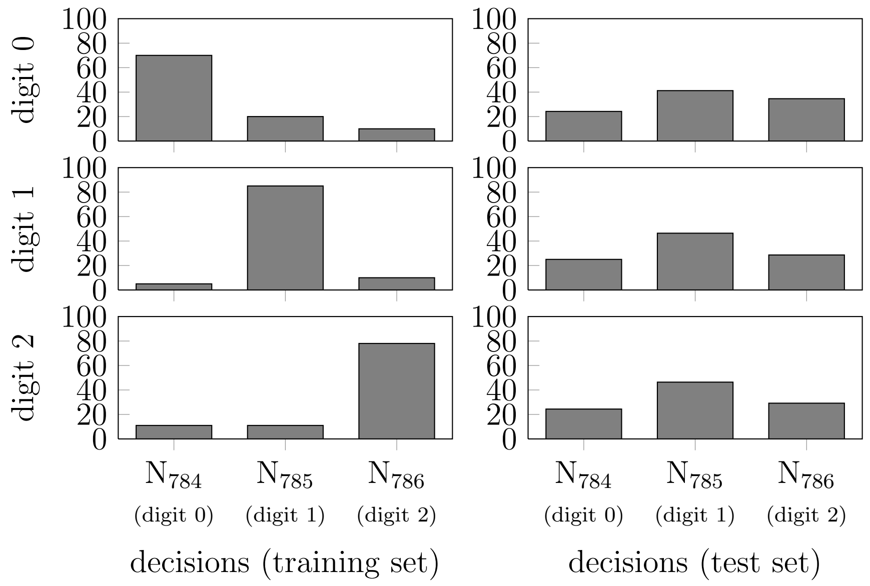


Figure 4.6: Bars show how often which decision was made for each digit in percentage. The first row holds the decisions when digit 0 is presented, the second row if 1 is presented and the third row if 2 is presented. For a training set (left) and a test set (right) from the MNIST database, pixel gray values were encoded as spike rates.

Fig. 4.5 shows the recognition rate while training the network. It is plotted by storing if the decision is true or false for each epoch in a vector and smooth this vector with a moving average, for further informations see section B.7. This curve gives an indication if the learning process was successful or not.

In this example the large training set is learned well. Around 80% of the training examples can be recognized correctly after the learning process (4.6, left). Even if the relatively large training set can be learned well, the generalization with this network fails. In fig. 4.6 (right) the decisions which are made while presenting the test set to the network are displayed. There is no general learning success ascertainable.

The weights are moving together during the learning process (compare fig. 4.8 top and bottom), however the mean value keeps approximately constant. Among themselves the weights remain equably distributed (a real uniform distribution can no longer be assumed), but after the learning other connections have the strongest weights. This can be gathered from figure A.7 in the appendix, too. Evenly distributed weights become apparent as an indicator for the quality of the learning process. The results of another simulation are displayed in the appendix (sec. A.1). Here the ratio of η_-/η_+ is chosen smaller (0.5 instead of 2.5), thus the positive learning rate have a stronger influences and most of the weights are increased. A decreasing of lesser important connection weights does not take place sufficiently. The result is correspondingly deteriorated.

The receptive field of output neuron N_{786} (digit 2) in figure 4.7 is taken from another simulation, but with the same parameters used in this chapter. Even after many training epochs an arising of dedicated patterns is not visible. This could reflect the missing ability of generalization. A reliable statement of what and especially how the network learns the given training set can not be given here.

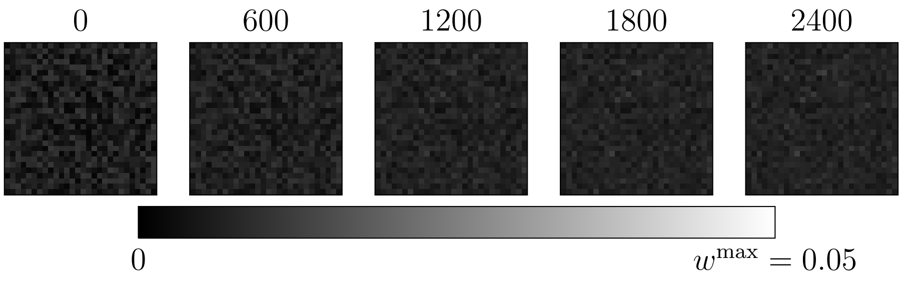
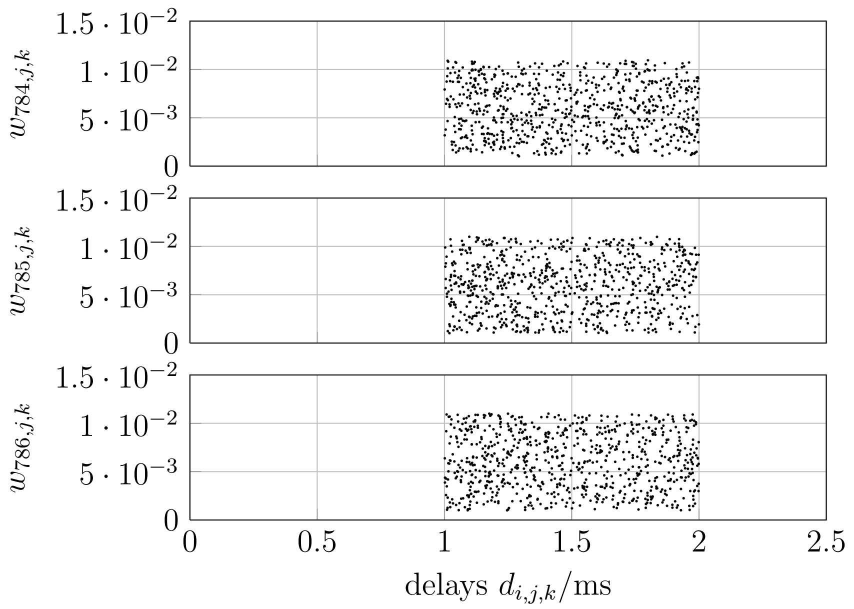


Figure 4.7: Receptive field of output neuron N_{784} (digit 2) in different training epochs.



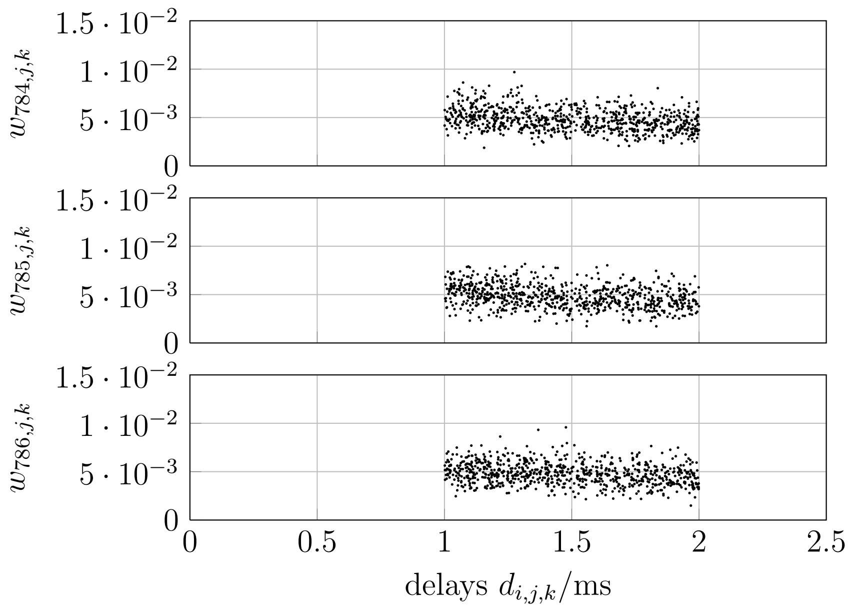


Figure 4.8: Connections leading to the three output neurons before (top) and after (bottom) the learning process. Weights are plotted over related delays, each mark represents one connection.

4.3.2 Pixel gray values encoded as time-to-first-spike

In fig. 4.9 one sees the activity heatmap of the input layer. A two is represented, thus each neuron fires once according to the time-to-first-spike coding (sec. 4.1). A first spike wave occurs at 10ms. These are the neurons with the written digit and a white image gray value of 255. In the following some individual spikes appear in the surrounding of the written digit. These are some darker gray values in the lateral areas of the digit. At 90ms the pixels of the black background are spiking. Because spike rates are displayed, a single spike leads to a rate of $1/\Delta t = 1/(5\text{ms}) = 200\text{Hz}$ in the figure.

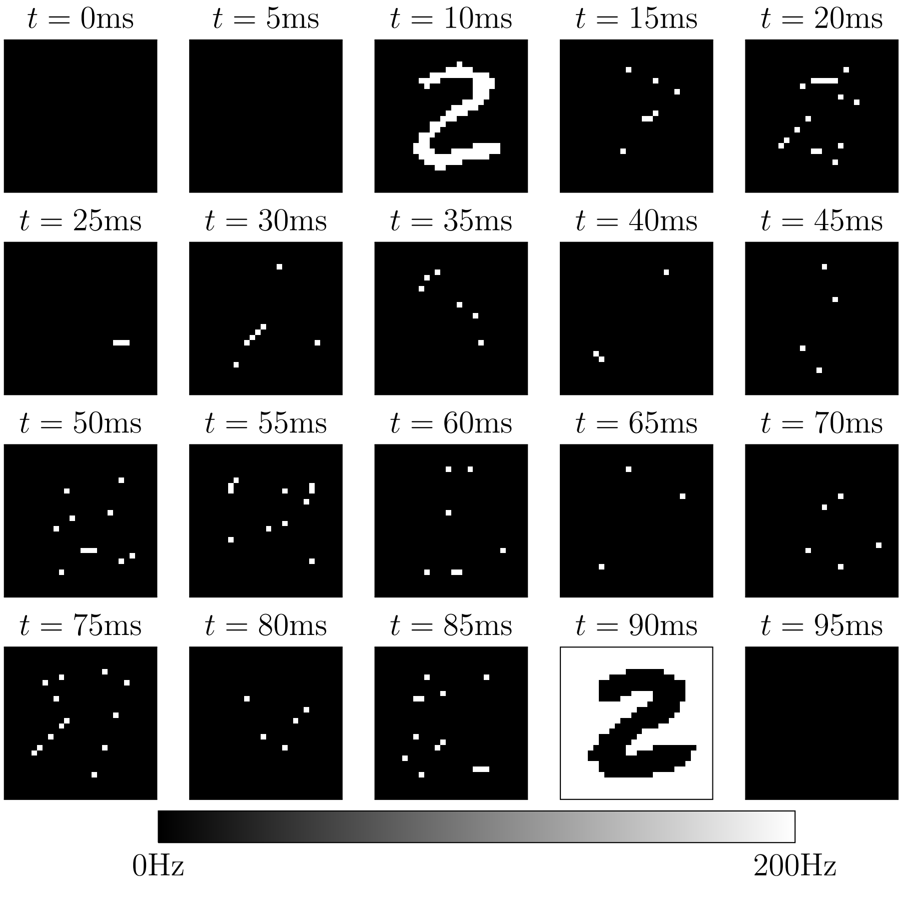


Figure 4.9: Exemplary activity heatmap of the input layer with 28^2 neurons and a handwritten "two" of the MNIST database. Spikes are generated according to equation 4.2, $t_{\text{min}} = 10\text{ms}$, $t_{\text{max}} = 90\text{ms}$, $p_{\text{max}} = 255$. In the heatmap black corresponds to $\nu = 0$ and white to $\nu = 200\text{Hz}$. Moving average parameters: integration window $\Delta t = 5\text{ms}$, discretization step width $\mathrm{d}\,t = 1\text{ms}$.

Because each input neuron is connected to each neuron of the consecutive layer, all 28^2 spikes will arrive in the output layer, too. Fig. 4.10 shows the input and output spikes of the three output neurons. A first spike wave arrives at the neurons between 11ms and 19ms. These are the spikes of the input neurons firing at 10ms. Caused by the uniformly distributed delays ($d_{i,j} = 1 \ldots 9$ms) the wave is temporally spread. After that the spikes of the lateral areas of the digit are following and then those of the background.

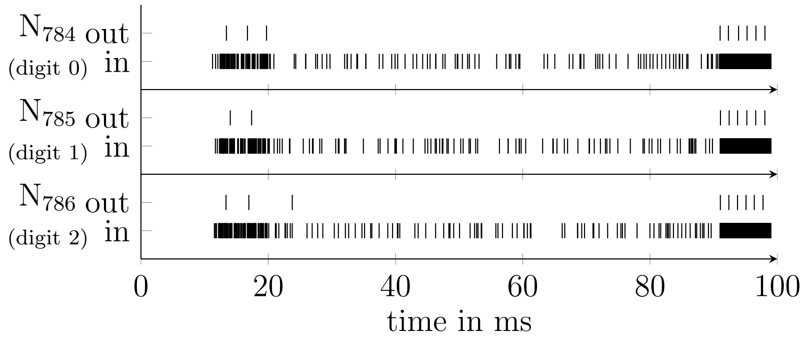


Figure 4.10: Exemplary spike trains of the three output neurons N_{784}, N_{785} and N_{786}. The first input spike wave ($11 \ldots 19$ms) is caused by the white areas in the input patterns, the written digits. The second wave ($92 \ldots 100$ms) is caused by the background. Differences in the spike patterns arise through different connection delay times.

All connection weights are initialized very close to each other, within the interval $[0.04, 0.041]$. The LIF neurons in the output layer ($\Delta_{\text{abs}} = 1$ms, $\tau_{\text{m}} = 15$ms) are responding. If the winner neuron is selected by the first spike times ($t^{(0)}_{784} = 13.434$m, $t^{(0)}_{785} = 14.037$m and $t^{(0)}_{786} = 13.364$m in fig. 4.10), the winner in this case is neuron N_{786}. This is the correct decision, because a two has been presented. Time-to-first-spike decoding provides an unique winner in all probability. A simultaneous spiking is unlikely, because of the numerousness of different delays.

After the initialization the network is trained with the STDP algorithm for 3000 epochs. A test set with 3000 examples (1000 times digit 0, 1000 times digit 1, 1000 times digit 2) and a training set with 300 examples (100 times digit 0, 100 times digit 1, 100 times digit 2) is used. Parameters of the learning algorithm are $\eta_+ = 0.02$, $\eta_- = 0.6 \cdot \eta_+ = 0.012$, $\tau_+ = \tau_- = 11$ms and $w^{\max} = 0.1$. Only the first spike of the real winner is taken as reference spike for the learning process within one epoch.

Here a relatively small learning rate is chosen. Larger learning rates ($\eta_+ = 0.2$, $\eta_- = 0.12$) are working well for small training sets. For larger training sets the network adapts too fast and cannot learn in consequence. Their ratio is more important than the magnitude of the learning rates. A ratio of $\eta_-/\eta_+ = 0.6$ between the two learning rates has turned out to be good.

Fig. 4.11 shows the recognition rate while training the network. It reveals how the recognition rate increases or the number of wrong decision decreases respectively over several epochs. Compared with the training run in section A.3 where a significantly smaller training set is used, the recognition rate is smaller on average and the curve is more noisy after several epochs. However, the recognized number of digits of the training set after the learning and the possibility of generalization cannot be seen from this figure.

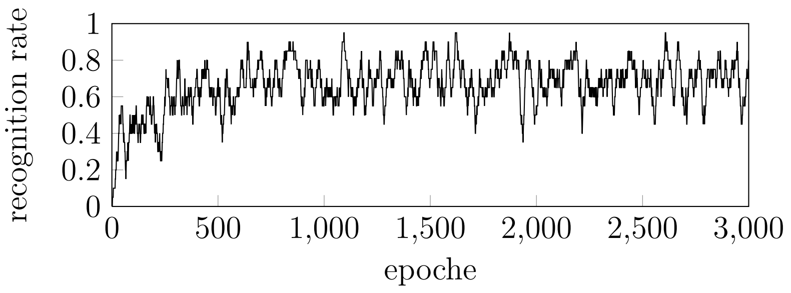


Figure 4.11: Learning process of a training set from the MNIST database, pixel gray values are encoded with time-to-first-spike.

To get an overview of how well the training set is learned, all 300 examples from the training set are presented once again without calling the STDP algorithm. The results are displayed in fig. 4.12 on the left. The digits seem to be learned passably good, in any case there is a tendency to a correct recognition. In order to prove the ability to identify examples which are not part of the training set, the test set is presented. The results of this generalization test are displayed on the right in fig. 4.12. One can see that the result are only slightly below and very similar to the result with the training set. This indicates that generalization is possible, but of course within the bounds of quality of learning on the training set. In comparison with the example in the appendix (sec. A.3) one can observe the results for a small training set. Here it may be essential that the number of connections is large and the basis of information is relatively small. The training set is learned very well, but because of the small amount of data generalization fails.

Having a closer look the to the weights before (fig. 4.13 top) and after (fig. 4.13 bottom) the learning process, one sees that some weights are changed

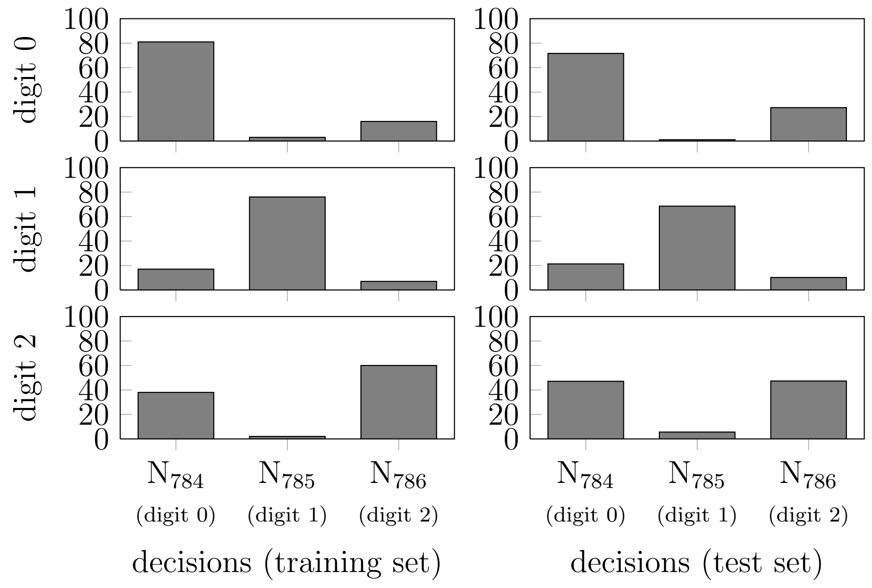
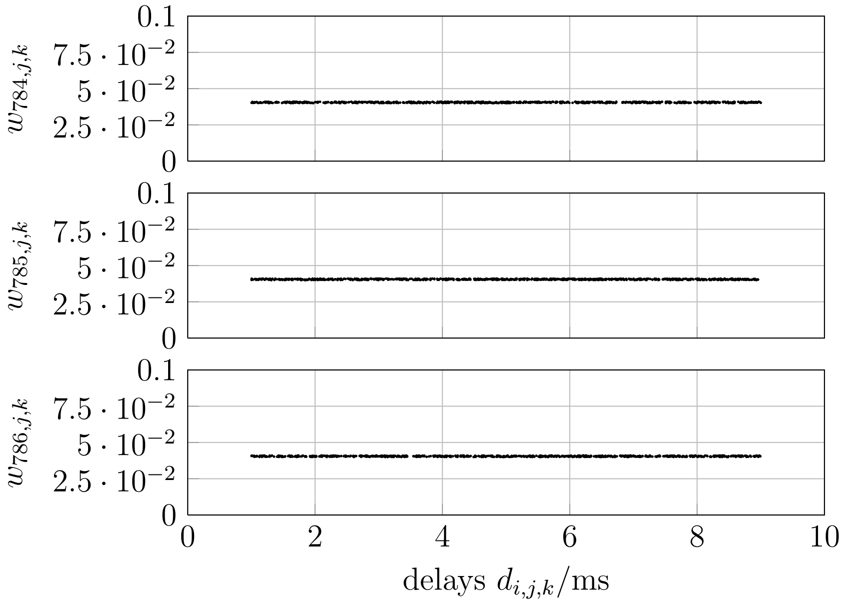


Figure 4.12: Bars show how often which decision was made for each digit in percentage. The first row holds the decisions when digit 0 is presented, the second row if 1 is presented and the third row if 2 is presented. For a training set (left) and a test set (right) from the MNIST database, pixel gray values were encoded as time-to-first-spike.

but a large number of the weights are unaltered or only slightly changed. This is also shown in another example from the appendix (fig. A.11). There a random chosen subset of the connection weights leading to the three output neurons is displayed. Some of the weights are running against specific values, which is the expected behaviour. But a large number of the weights are unaltered or only slightly changed. This could be an indicator of a too large number of connections. One aspect are the large image parts with background pixels. They have the same gray value (zero) and are therefore firing very late at $t = 90$ms. These pixels are not or only very little included in the learning process, because the output spikes in the output layer will be earlier and the STDP window is relatively small. This is fine because theses pixels do not carry any new information. Another aspect lies in the different delay times. When comparing figure 4.13 (top) and 4.13 (bottom) it can reasonably be concluded that the connections with greater delays are rarely increased. Only a few connection weights with short delays are strongly increased. Altogether negative changes are weaker. On the one hand this can be deduced by the smaller negative learning rate. On the other hand the weights are approaching the lower soft bound.



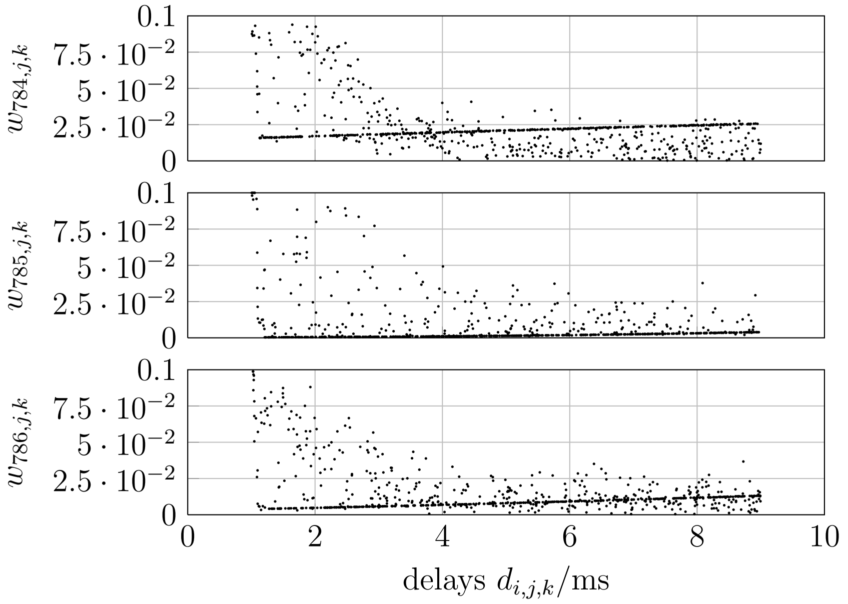


Figure 4.13: Connections leading to the three output neurons before (top) and after (bottom) the learning process. Weights are plotted over related delays, each mark represents one connection.

An elaboration for this issue gives figure 4.14. It is recognizable that due to the learning process especially connections which refer to areas that are part of the digits are altered. This is because those input neurons are firing earlier. The increasing of those connection weights seems to be the obviously needed behaviour. But it can also be observed that digit characteristically connection weights are strongly decreased, which is probably not very practically. Connection weights which refer to input neurons which are always representing background pixel are slowly decreased over the epochs.

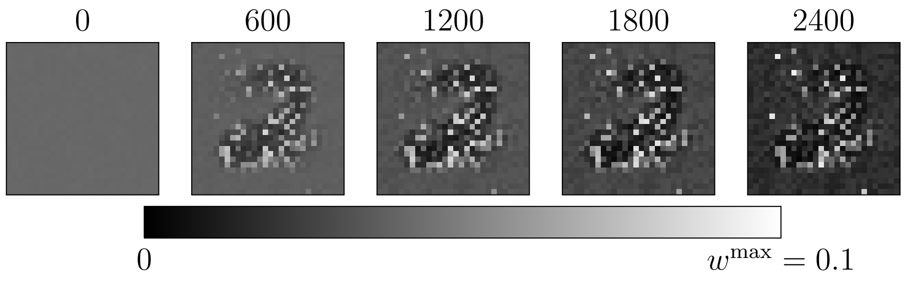


Figure 4.14: Receptive field of output neuron N_{784} (digit 2) in different training epochs.

Chapter 5

Conclusion

Within the scope of this thesis a wide ranging inside into the topic of SNNs was given. Due to limitations of time and the considerable subject a deeper analysis was not possible at some points in this application-oriented approach. Nevertheless, started form scratch by concerning the used neuron model up to a functioning application, a numerousness of aspects could be examined. Potentials and limitations of different approaches of neural coding were described in the beginning (chap. 2). Furthermore a new way of simulating aggregates of IF and LIF neurons was introduced in order to get an efficient simulation platform for the following investigations. The dualism of rate and temporal coding as well as the the role of neurons as coincidence detectors could be pointed out while studying simple neural circuits in chapter 3.

The analysis in chapter 4 led to some findings of how to parametrize the network and pointed out some principles of operation. The upper threshold of all neurons was the fix value $\vartheta_{\mathrm{u}} = 1$. This is possible because changing this parameter has the same effect as introducing a proportionality factor of the corresponding weights. The lower threshold ϑ_{l} is chosen lower or equal zero. Exact values are unimportant, because a negative pre-loading without inhibitions is not possible anyway. The membrane constants are chosen arbitrarily but within the range of biological neurons. Values are around $\tau_{\mathrm{m}} = 10 \ldots 25$ms. Values of the absolute refractory period of the input neurons should be chosen as high as they have no effect on the eigenfiring rates. Output neurons' refractory periods are of no importance, because the determination is done by the time-to-first-spike procedure.

To chose a reasonable initialization of the connection parameters was proved to be more difficult. Care must be taken on connection weights. They have to be be small enough to let several spikes arrive at the output neuron before a output spike occurs. Weights also have to be large enough so that the target neuron fires at all. Furthermore the initialization must be noisy to enable adequate differences between the connections in order to allow different decisions. On the other hand the noise amplitude must not reach a level on which the network always makes the same decision. Bearing in mind that several input spikes should be considered, one is operating on the very steep part of function 3.1. Thus a fine weight tuning for the initialization and especially for the learning is obligatory. In the case of IF neurons and rate encoded gray values weights are chosen between $w_{i,j} \in [1 \cdot 10^{-3}, 11 \cdot 10^{-3}]$, thus approximately 500 spikes are leading to an output spike. Because the spike trains' phases are normal distributed with a standard deviation of $\sigma = 5\text{ms}$ some input neurons fired twice before the output spike occurs. However, other neurons have not fired at all. In this way different pixels of the image can be addressed and assigned to the corresponding output neurons. The connection delays are of less influence. Noise with a certain degree is important to avoid simultaneous firings in the output layer, in order to make unique decisions.

In the case of LIF neurons and time-to-first-spike encoded gray values weights are chosen within the range $w_{i,j} \in [0.04, 0.041]$, thus approximately 25 simultaneously arriving spikes lead to an output spike. In fact this number is certainly larger, because spikes will not arrive precisely simultaneously. Connection delays with differences which are large enough to achieve the coincidence effects described in section 3.2 are important. Here values are chosen as $d_{i,j} \in [1\text{ms}, 9\text{ms}]$, which is in the range of the membrane time constant.

The STDP algorithm turned out as functional possibility to adapt the network's weights. Both in rate and in temporal encoded images it generates good results if the winner neuron is determined with time-to-first-spike. For this purpose the learning rule has to be applied to the first spike of the winner neuron's output spike train, otherwise a deterioration in the results is observable. To adjust the learning rates one has to consider the training set size. For large training sets, the ratio has to be smaller than for smaller training sets. Otherwise the weights are adjusted too fast and stable states cannot be reached.

For the success of a learning process the ratio of η_+/η_- is more important than the learning rates themselves. This ratio influences greatly if the weights are shifted up or down together. Clearly this should not happen. If it comes to a relatively equable distribution of the weights within the interval $(0, w^{\max})$ it seems to lead to better results. Hereby $w^{\max}$ avoids that single weights are in a range to generate an output spike alone. The ratio of the time parameters τ_+/τ_- seems to have a similar effect like the ratio of the learning rates. Furthermore they should be in the same range of the neurons time constant. If one is working with IF neurons the time constants have to be in the range of a few milliseconds. The options of rate decoding have not yet been finally clarified. The here used training and test examples did not achieve satisfying results.

In general it can be observed that rate encoded images in combination with IF neurons are learning the training set very well but the generalization fails. Maybe there is a chance of generalization if one increases the training set on a PC with a greater working memory. For temporal encoded images with LIF neurons the training results are a bit worse, but generalization is visible within the limits of the training success.

In future studies the difference in testing and training procedures between IF and LIF neurons may be an interesting subject. The deeper knowledge could lead to the possibility of initializing network and STDP parameters in order to enable a combination of coincidence and integration in an optimal way. This improvement could be supported by larger test series on a high-performance computer. It is reasonable to assume that due to optimisation further improvements are attainable, because small parameter changes yield significant effects. Another interesting issue should be the research of a STDP back-propagation algorithm in order to insert one or more hidden layers which may increase the performance. Furthermore the possibilities of a learning algorithm which adapts the connection delays may yield a performance and quality increase, too. In long term the possibilities of recurrent networks need to be the subject of further examination.

Regarding the present state of research (and what remains unclear) it should be possible to solve much more complex tasks. In conclusion SNNs have been and will remain an interesting research subject with great potentials.

Appendix A

Further MNIST recognitions

A.1 Rate encoding, bad ratio of learning rates

In this example the ratio $\eta_- / \eta_+ = 0.5$ is not optimal. Due to this all weights are increased while training and are not adjusted reasonable. The result is a large remaining error on the training set. In consequence the generalization fails, too.

The network is trained over 3000 epochs. The training set consists of 150 examples (50 times digit 0, 50 times digit 1, 50 times digit 2) and the test set consists of 450 example (150 times digit 0, 150 times digit 1, 150 times digit 2). Spikes are generated by rate encoding. The decoding in the output layer is done by time-to-first-spike.

Neuron parameters (IF): upper threshold $\vartheta_\mathrm{u} = 1$, lower threshold $\vartheta_\mathrm{l} = -0.1$, absolute refractory period $\Delta_\mathrm{abs} = 1\mathrm{ms}$.

Initialization of connections parameters between input and output layer, each output neuron is connected to each input neuron: weights $w_{i,j} \in [1 \cdot 10^{-3}, 11 \cdot 10^{-3}]$, delays $d_{i,j} \in [1\mathrm{ms}, 2\mathrm{ms}]$.

STDP parameters:

- Positive learning rate $\eta_+ = 2 \cdot 10^{-3}$
- Negative learning rate $\eta_- = 0.5 \cdot \eta_+ = 1 \cdot 10^{-3}$
- Time constants $\tau_+ = \tau_- = 5\mathrm{ms}$
- Weight maximum $w^{\max} = 0.05$
- Considering only first output spike

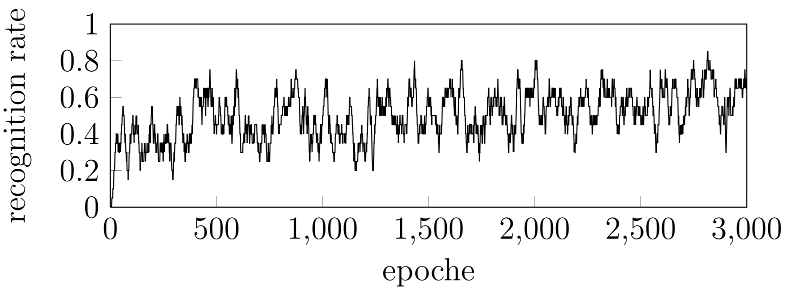


Figure A.1: Learning process of a training set from the MNIST database, pixel gray values are encoded as spike rates.

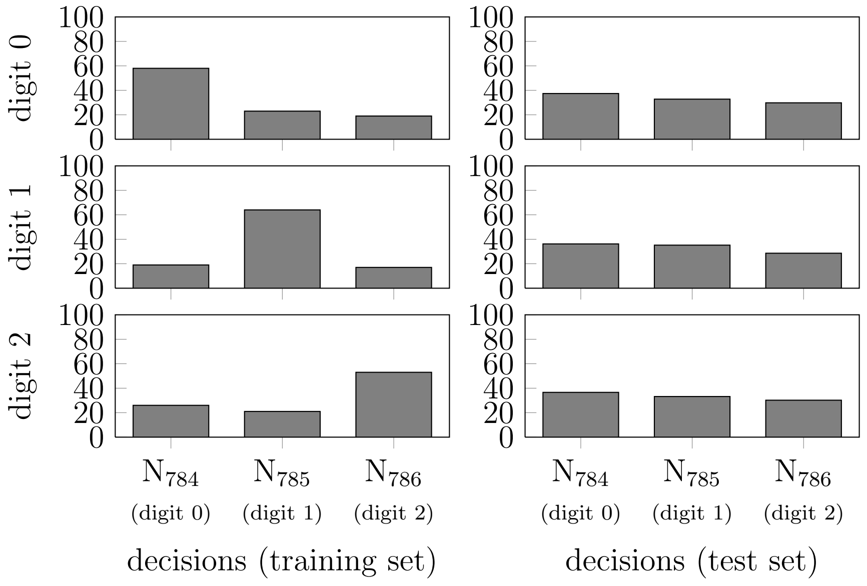


Figure A.2: Bars show how often which decision was made for each digit in percentage. The first row holds the decisions when digit 0 is presented, the second row if 1 is presented and the third row if 2 is presented. For a training set (left) and a test set (right) from the MNIST database, pixel gray values were encoded as spike rates.

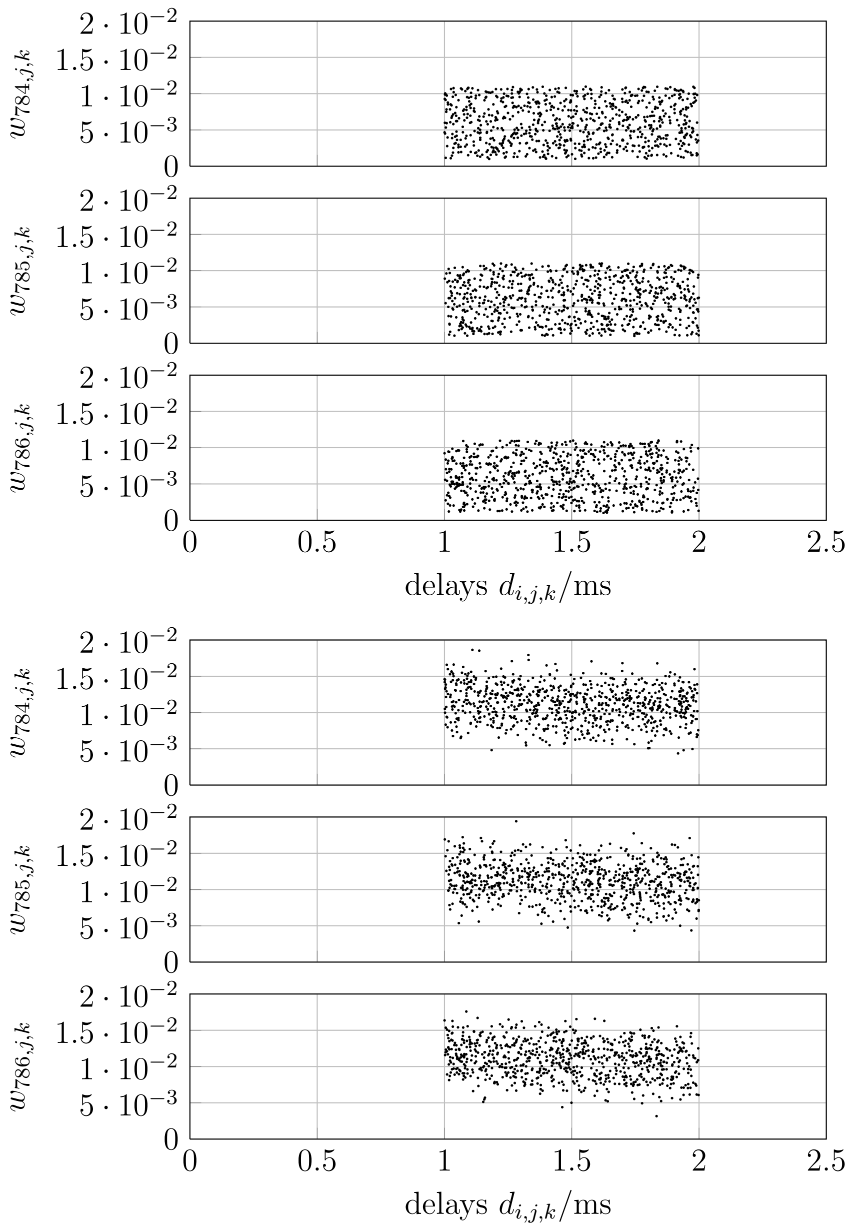


Figure A.3: Connections leading to the three output neurons before (top) and after (bottom) the learning process. Weights are plotted over related delays, each mark represents one connection.

A.2 Rate encoding, small training set

In this example the training set is learned well.

The network is trained over 300 epochs. The training set consists of 15 examples (5 times digit 0, 5 times digit 1, 5 times digit 2) and the test set consists of 150 example (50 times digit 0, 50 times digit 1, 50 times digit 2). Spikes are generated by rate encoding. The decoding in the output layer is done by time-to-first-spike.

Neuron parameters (IF):

- Upper threshold $\vartheta_{\mathrm{u}} = 1$
- Lower threshold $\vartheta_{\mathrm{l}} = -0.1$
- Absolute refractory period $\Delta_{\mathrm{abs}} = 1\mathrm{ms}$

Initialization of connections parameters between input and output layer, each output neuron is connected to each input neuron:

- Weights $w_{i,j} \in [1 \cdot 10^{-3}, 11 \cdot 10^{-3}]$
- Delays $d_{i,j} \in [1\mathrm{ms}, 2\mathrm{ms}]$

STDP parameters:

- Positive learning rate $\eta_{+} = 2 \cdot 10^{-3}$
- Negative learning rate $\eta_{-} = 2.5 \cdot \eta_{+} = 5 \cdot 10^{-3}$
- Time constants $\tau_{+} = \tau_{-} = 5\mathrm{ms}$
- Weight maximum $w^{\max} = 0.05$
- Considering only first output spike

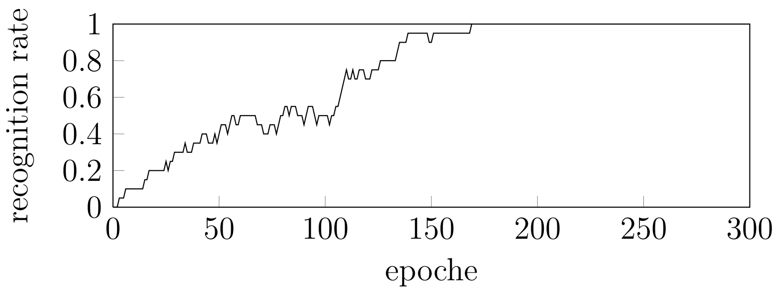


Figure A.4: Learning process of a small training set from the MNIST database, pixel gray values are encoded as spike rates.

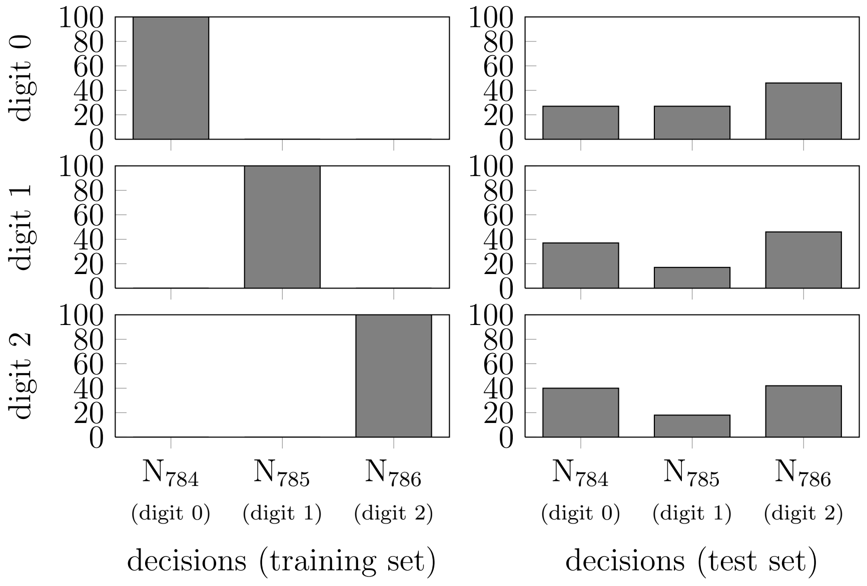


Figure A.5: Bars show how often which decision was made for each digit in percentage. The first row holds the decisions when digit 0 is presented, the second row if 1 is presented and the third row if 2 is presented. For a training set (left) and a test set (right) from the MNIST database, pixel gray values were encoded as spike rates.

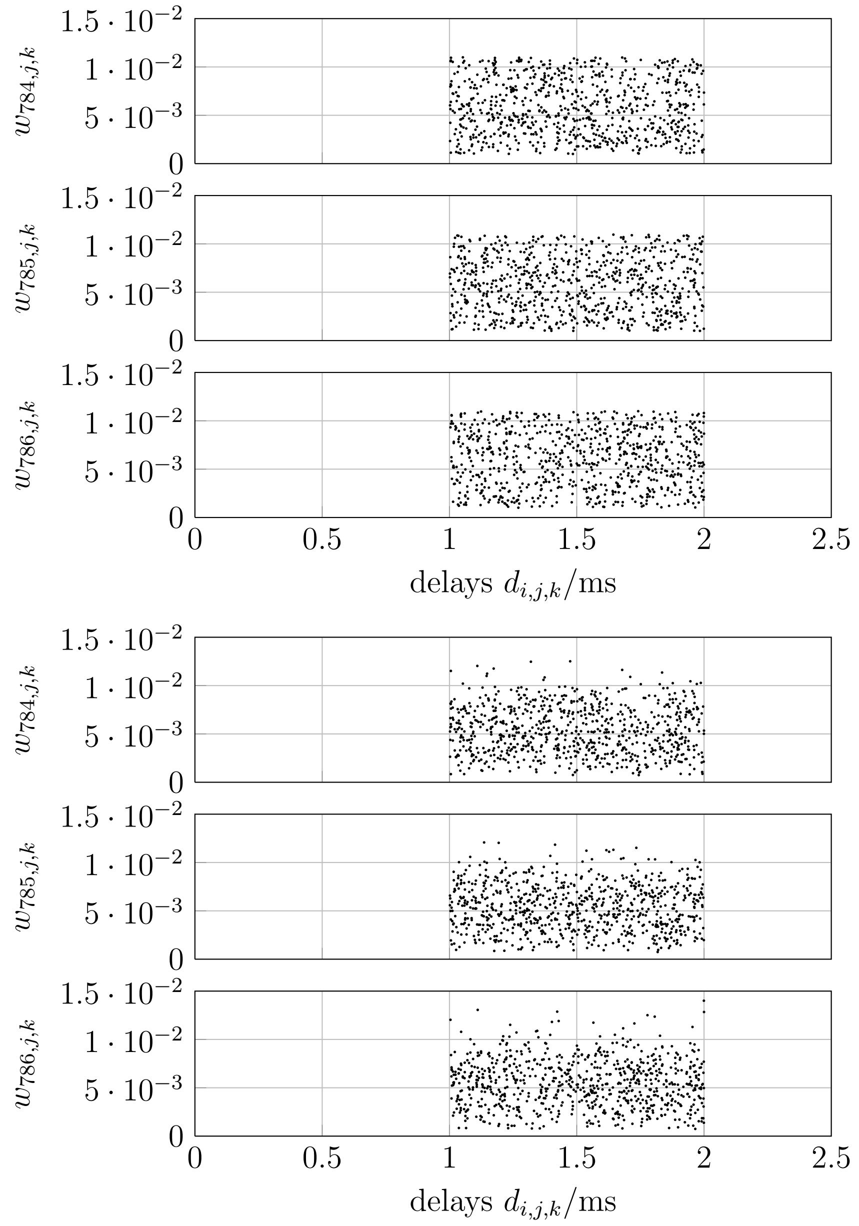


Figure A.6: Connections leading to the three output neurons before (top) and after (bottom) the learning process. Weights are plotted over related delays, each mark represents one connection.

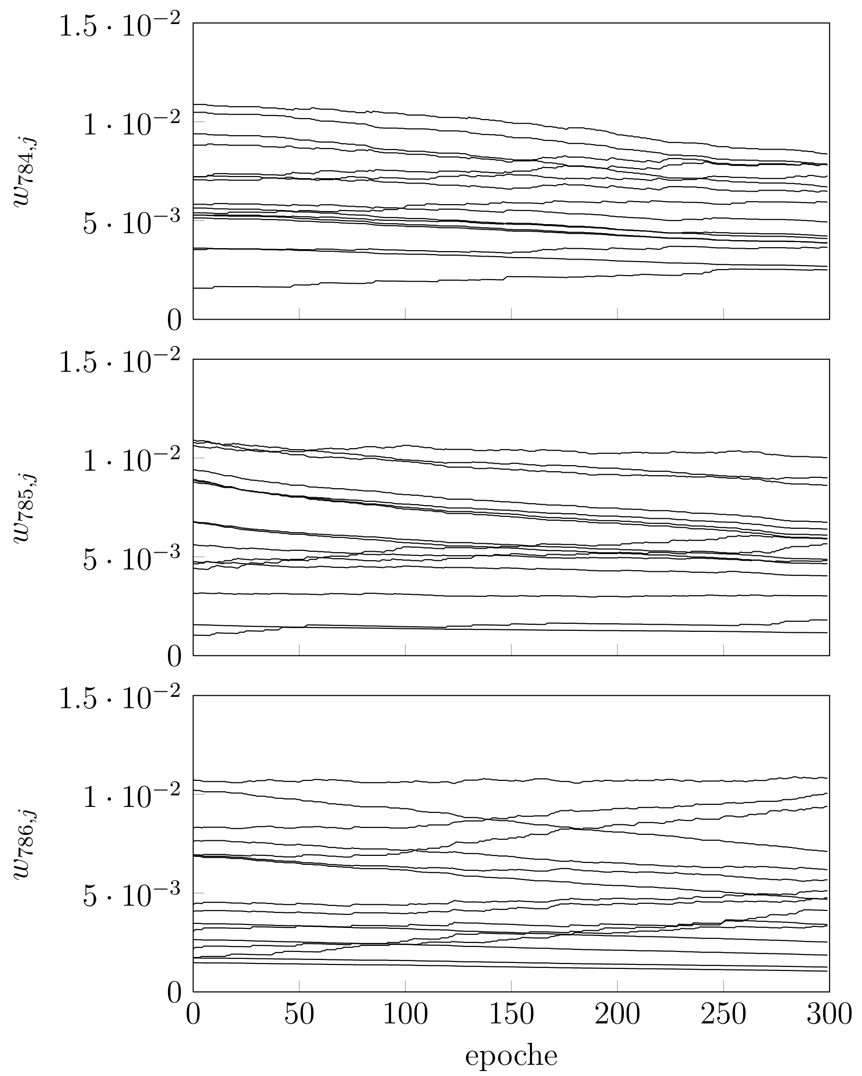


Figure A.7: A random chosen subset of the connection weights leading to the three output neurons over the learning epochs. The training set is relatively small.

A.3 Time-to-first-spike encoding, small training set

A network with one input and one output layer, similar to network in section 4.3.2. But here the network is only trained over 300 epochs and the training and test set size is reduced. A generalization is not visible. The training set consists of 15 examples (5 times digit 0, 5 times digit 1, 5 times digit 2) and the test set consists of 300 example (100 times digit 0, 100 times digit 1, 100 times digit 2). Spikes are generated by time-to-first-spike encoding. The decoding in the output layer is done by time-to-first-spike, too.

Neuron parameters:

- Upper threshold $\vartheta_{\mathrm{u}} = 1$
- Lower threshold $\vartheta_{\mathrm{l}} = -0.1$
- Membrane time constant $\tau_{\mathrm{m}} = 15\mathrm{ms}$
- Absolute refractory period $\Delta_{\mathrm{abs}} = 1\mathrm{ms}$

Initialization of connections parameters between input and output layer, each output neuron is connected to each input neuron:

- Weights $w_{i,j} \in [0.04, 0.041]$
- Delays $d_{i,j} \in [1\mathrm{ms}, 9\mathrm{ms}]$

STDP parameters:

- Positive learning rate $\eta_{+} = 0.02$
- Negative learning rate $\eta_{-} = 0.6 \cdot \eta_{+} = 0.012$
- Time constants $\tau_{+} = \tau_{-} = 11\mathrm{ms}$
- Weight maximum $w^{\max} = 0.1$
- Considering only first output spike

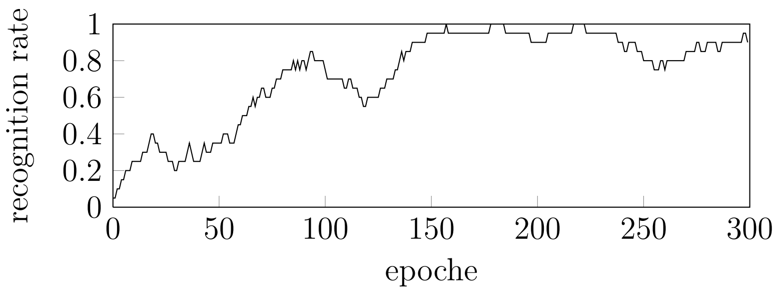


Figure A.8: Learning process of a small training set from the MNIST database, pixel gray values are encoded as time-to-first-spike.

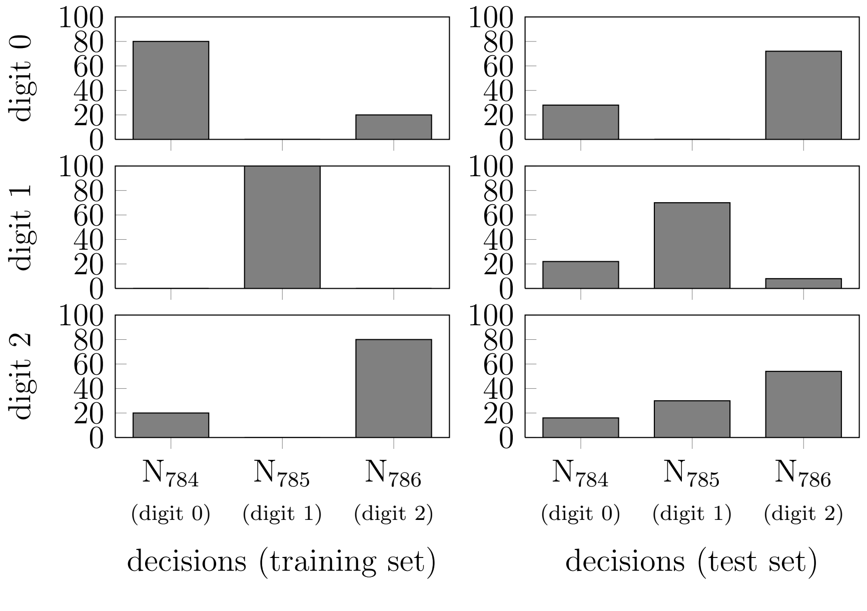


Figure A.9: Bars show how often which decision was made for each digit in percentage. The first row holds the decisions when digit 0 is presented, the second row if 1 is presented and the third row if 2 is presented. For a training set (left) and a test set (right) from the MNIST database, pixel gray values were encoded as time-to-first-spike.

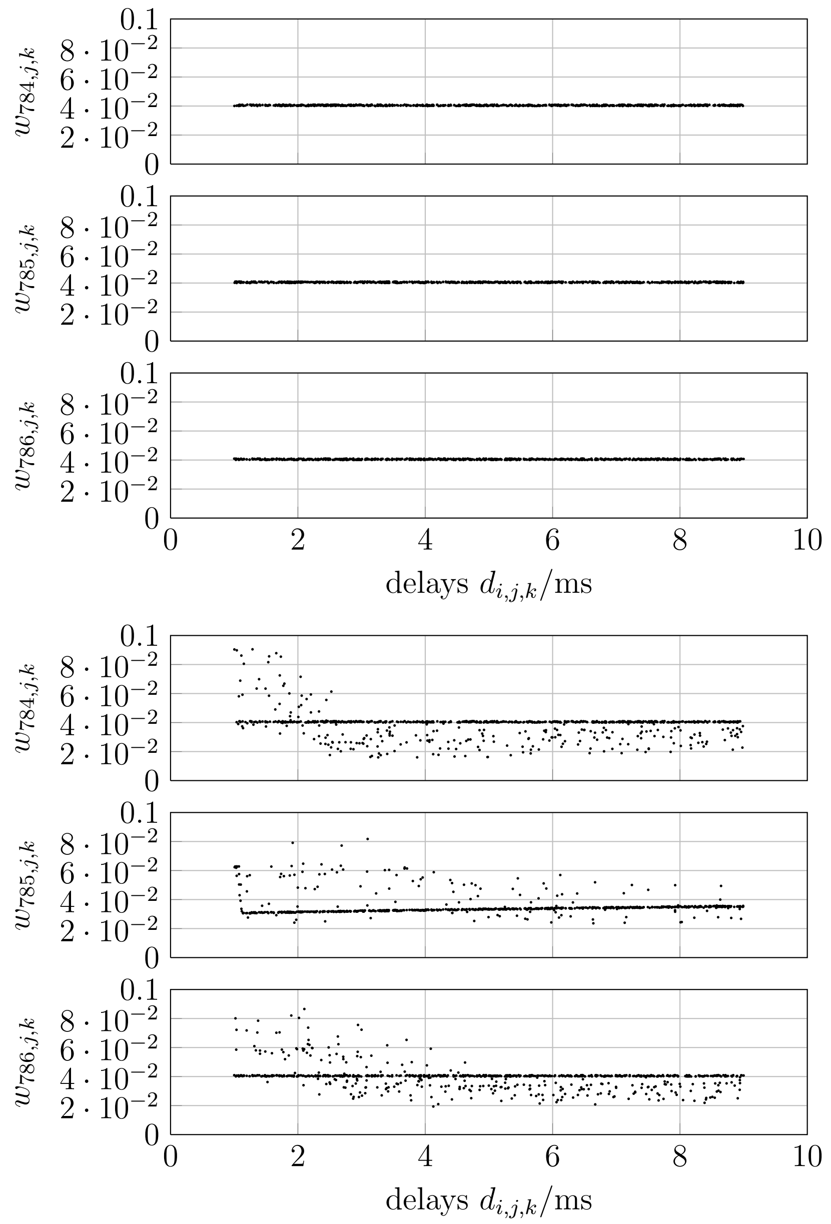


Figure A.10: Connections before learning begins. Each mark represents one connection.

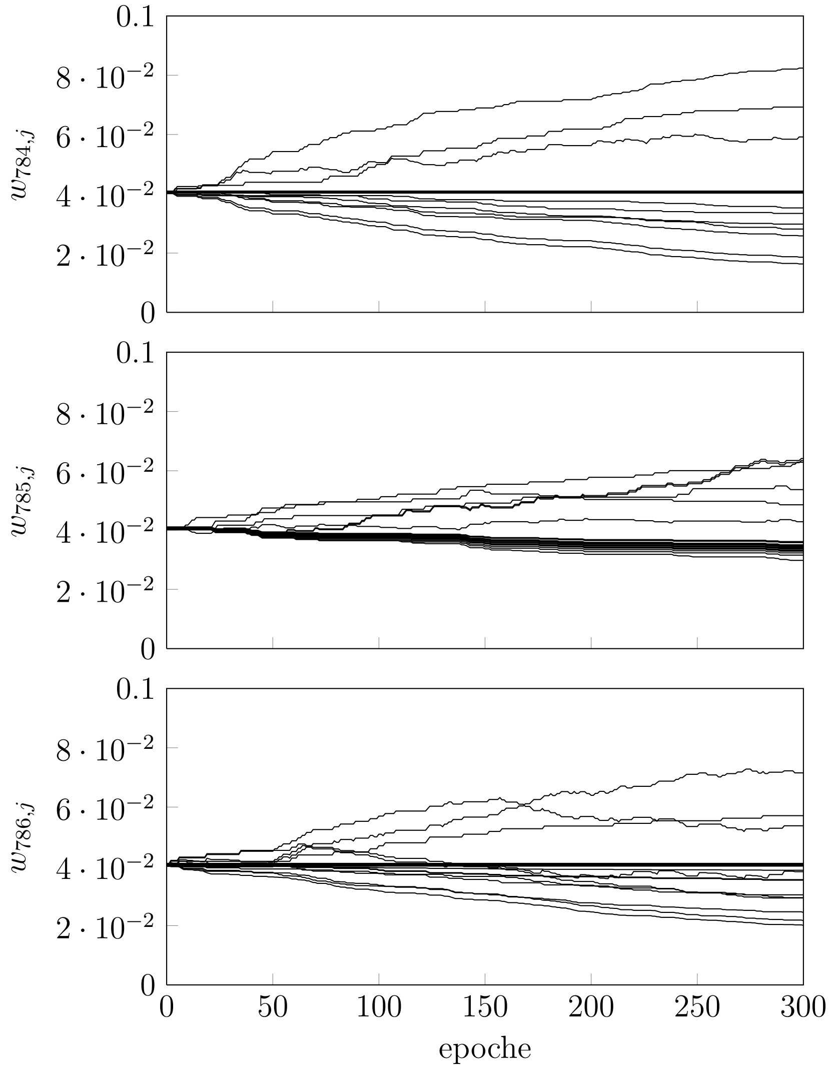


Figure A.11: A random chosen subset of the connection weights leading to the three output neurons over the learning epochs.

Appendix B

Implementation in C++

In the following a brief summary of the classes of the programmed SNN software is given. Especially the meanings of the parameters are summarized. Weight (tWeight) and time (tTime) is represented by double values, the time base is in ms.

KDevlop4 was used as IDE, to ease portability to other IDEs a cmake file was written, this can be found within the src-folder. The source code is documented with doxygen.

B.1 Network

This is the central class of the SNN project. It contains a vector which contains all neurons and one vector which contains all connections. The typical usage of the functions is in this order:

- *load(...)*;
- *initialize(...)*;
- *run(...)*;
- *learn(...)*;

The way to create a set of neurons and connections is to insert one vector which is containing all *Neuron::Parameter* and one vector containing all *Connection::Paramter* to the *Network::load()* function.

After all neurons and connection are loaded the network must be initialized with some eigenspikes. Therefore a vector with the *tSpikeSet*s is given to the initialize function.

To run the simulation for one epoch one has to call the *network::run()* function. A point of time when the simulation is interrupted must be handed over.

After simulating one epoch the network may be is supposed to learn. Therefore the network provides the opportunity to learn with STDP. Internally this function calls the learning function of each neuron given by the parameter *neuronIds*. To reduce calculation time the STDP learning window is bounded to the interval $[t_i^{(\mathrm{ref})} - 2 \cdot \tau_+, t_i^{(\mathrm{ref})} + 2 \cdot \tau_-]$, hereby $t_i^{(\mathrm{ref})}$ is the reference spike time. The STDP parameters are:

- *tUint neuronIds*
  Neurons whose afferent connections are included in the learning process.
- *double etaPos*
  Positive learning rate.
- *double tauPos*
  Time constant of positive interval of the exponential learning window.
- *double etaNeg*
  Negative learning rate.
- *double tauNeg*
  Time constant of negative interval of the exponential learning window.
- *double wMax*
  Upper bounds of weights. Lower bounds are zero.
- *bool onlyFirstSpike*
  Just the first spike of a given neurons spike train is considered if true.
- *string bounds*
  Defines if bounds are "soft-linear" or "hard". Default is "soft-linear".

B.2 Neuron

This class represents a neuron. The input spikes of each neuron are stored in a container at each neuron. Spikes within the containers are ordered by their time and spikes with equal spike times are added up. In contrast to one central spike container a decentralised storing has got the advantage of faster sorting of new spikes. With increasing network activity the number of spikes can increase immensely, thus spike sorting takes a major part of calculation time.

It would be less memory consuming to store the spikes as output spikes at the belonging neuron and to read this storage when calling the simulation function of the successor neurons. The way to store several copies of the output spikes (modified with the connection delay) as input spikes at the receiving neurons has got the advantage that the relevant spikes have not to be collected together from a large number of possibly extensive spike containers. A comparison of simulation times of both approaches has not been carried out. In order to initiate the network there must be a possibility to evoke output spikes without external stimulation with spikes. This kind of output spike is called eigenspike here. Eigenspikes are stored in the same container as the input spikes, but are marked by an invalid connection id, for example *UINT_MAX*.

Connections have to be registered, therefore pointers to all connections of a neuron are stored. Spikes are automatically delivered while simulating. How far the simulation can go is calculated at each time the *simulate()* function is called. Each neuron has got the following parameters.

- *tUint id*
  Id of this neuron, must be unique. Defaults to UINT_MAX.

- *tUint layerId*
  Layer this neuron belongs to. Defaults to UINT_MAX.

- *tWeight activationLowerBound*
  Lower membrane threshold, must be ≤ 0. Defaults to 0.

- *tWeight activationUpperBound*
  Upper membrane threshold, must be > 0. Defaults to 1.

- *tTime refractoryPeriod*
  Absolute refractory period, must be ≥ 0. Defaults to 0.001.

- *tTime leaky*
  Membrane time constant, disabled if ≤ 0. Defaults to -1.

B.3 Connection

To come into effect, the connections have to be notified at the neurons they are belonging to. Delays are always positive. Weights are positive if the connection is excitatory and negative if the connection is inhibitory. The boolean value *inhibiting* is necessary to define if the connection is excitatory or inhibitory when the weight becomes zero. Changes between inhibitory and excitatory connections are not in scope.

- *tUint id*
  Each connection has got its own id. Default is $UINT\_MAX$.
- *tUint originNeuronId*
  Id of the pre-synaptic neuron. Default is $UINT\_MAX$.
- *tUint targetNeuronId*
  Id of the post-synaptic neuron. Default is $UINT\_MAX$.
- *bool inhibiting*
  Connection is excitatory if false, inhibitory if true. Default is false.
- *tweight weight*
  Connection's weight, weight ≥ 0 if excitatory, weight ≤ 0 if inhibitory.
- *tTime delay*
  Connection's delay (delay > 0), time until a spike arrives at the post-synaptic neuron. Defaults to 1 ms.

B.4 Spikes

Each action potential is represented by one instance of this class. This class seems to be slightly oversized at a first sight. A spike receiving neuron requires information about when and with which strength a spike arrives. This information could be stored either as weight value or as a reference to the connection the spike takes. Furthermore the possibility of simultaneously arriving spikes must be handled, this means a summation of the belonging connection weights or a storing of their references. To avoid calling the connection instances several times the weights are stored at the spikes, but the connection IDs are stored as well to support the STDP algorithm. The boolean variable *evokedSpike* is only for the purpose of evaluation, it holds the information if an input spike is responsible for an output spike.

The *CompAndAddIfEqual* functor allows to sort spikes in a set and combine simultaneous arriving spikes by adding their weights. Spike parameters are:

- *tUint connectionId*
  Id of the connection this spike is transmitted with, more than one id if several spikes arrive at the same time.

- *tWeight weight*
  Sum of weights of the connections this spike is transmitted with.
- *tTime time*
  Time of arrival at the target neuron.
- *bool evokedSpike*
  True if this spike evoked an output spike. Defaults to false.

B.5 Network generator

The class *Networkgenerator* generates initial values of a network's neuron and connection parameters. The generation process starts by calling the *Networkgenerator::generate(...)* function. There are two types of parameters. The first type defines the attributes of a layer's neurons and the second type defines how these layers are connected with each other. Both parameter types can be defined in csv files and loaded with the Database.
The generated Neuron::Parameter and Connection::Parameter are returned by the corresponding getter functions.

B.5.1 Layer generation parameters

Each layer is represented by one object of this class, the parameters are:

- *tUint numberOfNeurons*
  Number of neurons within this layer.
- *tWeight activationLowerBound*
  Lower threshold of the neurons.
- *double activationLowerBoundJitter*
  Adds Gaussian noise to the lower threshold with this standard deviation.
- *tWeight activationUpperBound*
  Upper threshold of the neurons.
- *double activationUpperBoundJitter*
  Adds gausian noise to the upper threshold with this standard deviation.
- *tTime refractoryPeriod*
  Absolute refractory period of the neurons.

- *double refractoryPeriodJitter*
  Adds gausian noise to the refractory period with this standard deviation.
- *tTime leaky*
  Membrane time constant, no decay if zero or negative.

B.5.2 Connection generation parameters

Each group of connections is represented by one object of this class. It is possible to connect neurons via several connections. The parameters are:

- *tUint layerA*
  Layer of predecessor neurons.
- *tUint layerB*
  Layer of successor neurons.
- *tWeight weightsLowerBound*
  Weights are uniform distributed between weightsLowerBound and weightsUpperBound.
- *tWeight weightsUpperBound*
  Weights are uniform distributed between weightsLowerBound and weightsUpperBound.
- *tTime delaysLowerBound*
  Delays are uniform distributed between delaysLowerBound and delaysUpperBound.
- *tTime delaysUpperBound*
  Delays are uniform distributed between delaysLowerBound and delaysUpperBound.
- *double connectionProbability*
  The probability that a connection is established.
- *double inhibitorySynapsesRatio*
  The probability a connection is excitatory or inhibitory.
- *bool upperTriangle*
  A connection between pre-synaptic neuron j and post-synaptic neuron i can be established if $j < i$.

- *bool lowerTriangle*
  A connection between pre-synaptic neuron j and post-synaptic neuron i can be established if $j > i$.

- *bool mainDiagonal*
  A connection between pre-synaptic neuron j and post-synaptic neuron i can be established if $i = j$.

B.6 Spike generator

The class *Spikegenerator* generates the network's neurons eigenspikes. It is necessary to pass the total number of neurons to its constructor. The generation process starts by calling the *Spikegenerator::generate(...)* function. Several parameters are available which can be called one after another, the spikes sets are joined. The generated spike trains are returned by the function or accessed by a getter function.

The generated spike trains can be stored in csv files. Some of the generation parameters can be loaded from csv files, too.

All spike generator parameters are inherited from the *Spikegenerator::Parameter* object. The following general parameters are used:

- *tUint firstNeuronId*
  Id of the first neuron of the neuron-group spikes are generated for.

- *tUint lastNeuronId*
  Id of the last neuron of the neuron-group spikes are generated for.

- *tUint orginatorNeuronId*
  Eigenspikes have no predecessor neuron id. Therefore this typically should be an id that is not used yet.

- *tWeight weight*
  The weight of the spikes, typically 1 in order to always evoke an output spike.

- *double weightJitter*
  Adds Gaussian noise to the weights with this standard deviation.

- *tTime start*
  Beginning of the spike trains.

- *tTime end*
  Ending of the spike trains.

Equal spaced spikes

Parameter to generate a continuous firing rate:

- *tTime timeSpace*
  Time lag between two spikes, the same as $1/\nu$.
- *tTime phase*
  Phase of the spike train.
- *double phaseJitter*
  Adds Gaussian noise to the phase with this standard deviation.

Poisson spikes

Parameter to generate a poisson impulse process. The time lag between two spikes is exponentially distributed. The exponential distribution $p(x) = \lambda e^{-\lambda x}$ has got the single parameter *lambda*, which is the expectation value.

- *double lambda*
  Expectation value of the time lag between two spikes.
- *tTime phase*
  Phase of the spike train.
- *double phaseJitter*
  Adds Gaussian noise to the phase with this standard deviation.

Ramp spikes

Parameter to generate a linear frequency ramp. The slope is $(f_{\text{Max}} - f_{\text{Min}})/(t_{\text{end}} - t_{\text{start}})$.

- *double fMin*
  Start frequency of the ramp.
- *double fMax*
  Stop frequency of the ramp.

Gaussian spikes

Parameter to generate normal distributed spikes.

- *double my*
  Expectation value.

- *double sigma*
  Standard deviation of the normal distribution.

- *double numberOfSpikes*
  Number of normal distributed spikes. The number of spikes can differ. If some generated spikes are not within [start:end] (general parameters), they are ignored.

Image Single Parameter

A single spike is generated per pixel and neuron by transforming the gray value linearly into its time, see section 4.1.

- *tUint sx*
  Image size x (number of pixels).

- *tUint sy*
  Image size y (number of pixels).

- *tUint maxval*
  Max gray value.

- *vector< tUint > data*
  Vector with image data.

- *tTime tMin*
  Earliest spike time, corresponds to gray value of zero.

- *tTime tMax*
  Latest spike time, corresponds to max gray value.

Image Rate Parameter

A constant firing rate is generated per pixel and neuron by transforming the gray value linearly to a firing rate, see section 4.1. To avoid the same phase for all spike trains of an image, the phase is determined by a normal distribution with $\mu = 20$ms and standard deviation $\sigma = 5$ms.

- *tUint sx*
  Image size x (number of pixels).

- *tUint sy*
  Image size y (number of pixels).

- *tUint maxval*
  Max gray value.
- *vector< tUint > data*
  Vector with image data.
- *tTime isiMax*
  Maximal inter spike interval, corresponds to min gray value.
- *tTime isiMin*
  Minimal inter spike interval, corresponds to max gray value.

B.6.1 MNIST

The class *Mnist* provides the possibility to generate a test and a training set from the MNIST handwritten digit database. Therefore the above described *Spikegenerator::ImageSingleParameter* parameter is used to enable time-to-first-spike coding or the *Spikegenerator::ImageRateParameter* parameter is used to enable rate coding. A vector containing the digits which are used (this could be 0,1,...,9) has to be handed over, furthermore the number of training or test examples per used digit must be defined. The used digits are always taken from the beginning of the MNIST database.

B.6.2 Geometric

The class *Geometric* provides the possibility to generate a test and a training set from the four shapes from figure 4.1 with the above described *Spikegenerator::ImageSingleParameter* or *Spikegenerator::ImageRateParameter* parameters. The number of shapes can be chosen.

B.7 Evaluation

This class provides some evaluation functions. Next to the following display functions a function to determine a winner neuron of a given layer is implemented.

B.7.1 Display Crosscorrelation

Plots the crosscorrelation of two neurons' spike trains.

- *neuronIdA*
  Reference neuron.

- *neuronIdB*
  Target neuron.
- *binWidth*
  Temporal width of each bin.
- *numberOfBins*
  Number of bins.
- *start*
  No spikes considered before.
- *end*
  No spikes considered after.
- *output*
  Defines if gnuplot (”gplt”) or pdflatex (”latex”) output is generated. Defaults to ”gplt”.
- *filename*
  Filename. Defaults to ”cc”.

B.7.2 Display Spikes

Displays the spike trains of the given neurons in the defined interval. For each neuron are plotted input, output and eigenspikes (read from Database*).

- *neuronIds*
  Neurons whose spike trains shall be plotted.
- *Database**
  Database containing eigenspikes.
- *start*
  No spikes considered before.
- *end*
  No spikes considered after.
- *output*
  Defines if gnuplot (”gplt”) or pdflatex (”latex”) output is generated. Defaults to ”gplt”.
- *filename*
  Name of the output file. Defaults to ”spikes”.

B.7.3 Display Moving Average

Calculates and displays the moving average of given neurons' spike trains. Calculated by $\rho(t) = n_{\mathrm{K}}(t - \text{window}/2; \mathrm{t} + \text{window}/2)/\text{window}$ whereby n_{K} denotes the number of spikes within the interval $[t - \text{window}/2, \mathrm{t} + \text{window}/2]$.

- *neuronIds*
  Neurons of which moving average activities shall be displayed.
- *window*
  Width of sliding integration window used to calculate moving average.
- *dt*
  Discretisation step width.
- *start*
  No spikes considered before.
- *end*
  No spikes considered after.
- *output*
  Defines if gnuplot ("gplt") or pdflatex ("latex") output is generated. Defaults to "gplt".
- *filename*
  Filename. Default is "sma".

B.7.4 Display Moving Average Heatmap

Calculates the moving average for each neuron according to subsection B.7.3. Then the sma values are converted in to colors and displayed at several points of time.

- *neuronIds*
  Neurons of which moving average activities shall be displayed.
- *window*
  Width of sliding integration window used to calculate moving average.
- *dt*
  Discretisation step width.
- *start*
  No spikes considered before.

- *end*
  No spikes considered after.

- *sx*
  Number of pixel (neurons) in x direction (sx ≥ 2 and sx $\cdot$ sy == *neuronIds.size()*).

- *sy*
  Number of pixel (neurons) in y direction (sy ≥ 2 and sx $\cdot$ sy == *neuronIds.size()*).

- *times*
  Vector containing the times at which heat maps are created.

- *output*
  Defines if gnuplot (”gplt”) or pdflatex (”latex”) output is generated. Defaults to ”gplt”.

- *filename*
  Filename. Defaults to ”sma”.

B.7.5 Display Weights Vs. Delays

Displays weights over delays of the connections leading to the given neurons.

- *neuronIds*
  Considered neurons.

- *output*
  Defines if plotting with gnuplot (”gplt”) or pdflatex (”latex”). Defaults to ”gplt”.

- *filename*
  Filename. Defaults to ”wVsD”.

B.7.6 Weight Recorder

This is a subclass of *Evaluation* which enables the observation of weights over time. The ids of neurons whose afferent connections' weights are recorded must be handed to the constructor. Every time the function *WeightRecorder::record()* is called the momentary values are recorded. Results can be plotted with *WeightRecorder::display()*.

Furthermore the weight recorder gives the possibility to plot receptive fields in different epochs. The parameters of the *displayReceptiveField()* function are the following:

- *neuronId*
  Neuron which receptive field is plotted.
- *epochs*
  Epochs in which the receptive field is displayed.
- *sx*
  Number of pixel in x direction.
- *sy*
  Number of pixel in y direction.
- *maxWeight*
  Maximal weight value in plot, if negative maximum from all recorded weights is taken. Defaults to -1.
- *filename*
  Filename. Defaults to "receptiveField".
- *output*
  Defines if plotting with gnuplot ("gplt") or pdflatex ("latex"). Defaults to "gplt".

B.7.7 Performance Statistics

This subclass of *Evaluation* enables the recording and analysis of training successes while training a network. Each epoch the decision of the network and the correct decision have to be passed by the function *record()*.
The successes true or false are recorded and can be displayed smoothed with a moving average over time with *displaySuccesses()*.
How often which decision was made for each group of represented examples can be displayed with *displayWinnings()*.

B.8 Image

This is a helper class in order to handle images. It provides the possibility of pixel manipulation as well as writing PNG and PGM files.

B.9 Database

The class *Database* provides functionality for writing and storing different parameters. The here implemented version provides the reading and writing of comma separated files. It is possible to read vectors containing *Spikegenerator::EqualSpacedParameter*, *Spikegenerator::PoissonParameter*, *Spikegenerator::RampParameter*, *Spikegenerator::GaussianParameter* and *Networkgenerator::NeuronParameter*, *Networkgenerator::LayerParameter* and *Neuron::Parameter*, *Connection::Parameter* and *tSpikeSet*.

Furthermore it is possible to store vectors containing *Neuron::Parameter*, *Connection::Parameter* and *tSpikeSet*.

The parameters' members are stored according to the order of the descriptions in the preceding subsections. With the exception of *tSpikeSet*s each entry of a vector containing a parameter object is represented by one row in the text-files. Each entry in a vector which contains *tSpikeSet*s is stored in a particular file, whereby each row of each text file contains one spike.

Bibliography

E. D. Adrian, Y. Zotterman, *The Impulses Produced By Sensory Nerve Endings*, J. Physiol., 61(2), pp. 151-171, april 1926.

M. F. Bear, B. W. Connors, M. A. Paradiso, *Neurowissenschaften 'Ein grundlegendes Lehrbuch für Biologie, Medizin und Psychologie'*, Spektrum, 3. Auflage, 2008.

E. N. Brown, R. E. Kass, P. P. Mitra, *Multiple neural spike train data analysis: state-of-the-art and future challenges*, Nature Neuroscience, vol. 7, no. 5, pp. 456-461, may 2004.

K. Diesseroth, Lecture: *Personal Growth Series: Karl Diesseroth On Cracking The Neural Code*, Google Tech Talks, `http://www.youtube.com/watch?v=5SLdSbp6VjM`, november 2008.

W. Gerstner, W. Kistler, *Spiking Neuron Models 'Single Neurons, Populations, Plasticity'*, Cambridge University Press, 4th printing, ISBN 978-0-521-89079-3, 2008.

S. Grossberg, *Recurrent neural networks*, `http://www.scholarpedia.org/article/Recurrent_neural_networks`, 8(2):1888, 2013.

S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing, first edition, ISBN: 002352761-7 1949.

D. O. Hebb, *The Organization of Behavior*, New York: Wiley & Sons, 1949.

A. L. Hodgkin, A. F. Huxley, *A Quantitative Description Of Mambrane Current And Its Application To Conduction And Excitation*, J. Physiol., 117(4), pp. 500-544, august 1952.

E. M. Izhikevich, *Which Model To Use For Cortical Spiking Neurons?*, IEEE Transactions on Neural Networks, vol. 15, no. 5, pp. 1063-1070, september 2004.

K. Kirkland, *Analytical Tools*, Mulab, George Gerstein U. of Pennsylvania Neuroscience, `http://mulab.physiol.upenn.edu/analysis.html`, last update: july 2006.

R. Legenstein, C. Naeger, W. Maass, *What Can a Neuron Learn with Spike-Timing-Dependent Plasticity?* Neural Computation, 17, 23372382, 2005.

R. Mayrhofer, M. Affenzeller, H. Prähofer, G. Höfer, A. Fried, *DEVS Simulation of Spiking Neural Networks*, Cybernetics And Systems, 2,16, pp. 573-578, 2002.

J. Sjöström, W. Gerstner, *Spike-timing Dependent Plasticity*, `http://www.scholarpedia.org/article/Spike-timing_dependent_plasticity`, 5(2):1362, 2010.

S.Thorpe, D. Fize, C. Marlot, *Speed Of Processing In The Human Visual System*, Nature, 381(6582), pp. 520-522, june 1996.

S. J. Thorpe, A. Delorme, R. VanRullen, *Spike-based Strategies For Rapid Processing*, Neural Networks, 14(6-7), 715-726, 2001.

E. A. Wan, *Finite Impulse Response Neural Networks With Applications In Time Series Prediction*, Dissertation, Department of Electrical Engineering of Stanford University, November 1993.