RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Daniel Brüderle

Implementing Spike-Based Computation on a Hardware Perceptron

Diplomarbeit

HD-KIP-04-16

KIRCHHOFF-INSTITUT FÜR PHYSIK

Implementing Spike-Based Computation on a Hardware Perceptron

This diploma thesis has been carried out by Daniel Brüderle at the
Kirchhoff-Institute of Physics
University of Heidelberg
under the supervision of
Prof. Dr. Karlheinz Meier

Abstract

Implementing Spike-Based Computation on a Hardware Perceptron

This thesis presents the design, the implementation and the utilization of a device which simulates networks of spiking neurons on the basis of an already existing hardware Perceptron. For this purpose basic principles of information processing in biological and artificial neural networks are introduced. The functional extensions developed for the Perceptron are mainly based on the reconfiguration of a programmable array of logic gates that controls the network chip. Detailed technical descriptions of these additional devices are presented. A high-level software interface has been created that allows to make use of the developed simulation platform without knowing about hardware specific details. This software includes methods that map virtually generated three-dimensional neural networks to the available resources with a minimum of topological distortion.
The capability of the novel simulation system to perform rate-based and temporal information processing is shown. The successful execution of so-called liquid computing, which provides an anytime approximation of a target function, is described. With the applied methods, it was possible to keep but not to optimize the ability of the network to store information for a benchmark task. In further experiments the Perceptron and its extensions were configured to simulate a network of real cortical neurons. The benchmark problem was solved with this spike-based setup.

Implementierung von Informationsverarbeitung mit Aktionspotentialen auf der Grundlage eines Perzeptron-Chips

Diese Arbeit beschreibt den Entwurfsprozess, die Implementierung und die Benutzung eines Gerätes zur Simulation von aktionspotential-basierten neuronalen Netzen auf der Grundlage eines bereits vorliegenden Perzeptron-Chips. Zu diesem Zweck werden grundlegende Mechanismen der Informationsverarbeitung in biologischen und künstlichen neuronalen Netzen vorgestellt. Die funktionellen Erweiterungen, die für das Perzeptron entwickelt wurden, basieren im wesentlichen auf der Neuprogrammierung einer bereits vorhandenen rekonfigurierbaren Kontroll-Logik für den Chip. Die technischen Details dieser zusätzlichen Funktionseinheiten werden beschrieben. Es wurde eine Software entwickelt, die das Wissen um hardwarespezifische Besonderheiten unnötig macht und eine Nutzung des Systems mittels abstrahierter Schnittstellen ermöglicht. Diese Software beinhaltet unter anderem Methoden, um virtuelle dreidimensionale neuronale Netzwerke möglichst frei von topologischen Verzerrungen auf die zur Verfügung stehenden Ressourcen abzubilden.
Die Fähigkeit der neuartigen Simulationsumgebung zur raten- und intervall-basierten Informationsverarbeitung wird belegt. Die erfolgreiche Durchführung von sogenanntem Liquid Computing, einer Methode zur kontinuierlichen Approximation einer Zielfunktion, wird beschrieben. Die Fähigkeit eines Netzwerkes, Information zur Lösung eines Standardproblems zu speichern, konnte durch die zur Anwendung gekommenen Methoden erhalten, aber nicht verbessert werden. In weiteren Experimenten wurden das Perzeptron und seine Erweiterungen für die Simulation eines Netzes echter kortikaler Neuronen konfiguriert. Mit diesen Einstellungen wurde das ausgewählte Standardproblem aktionspotential-basiert gelöst.

Contents

Motivation

This thesis is the result of utilizing different ways of computation. It was written on a standard desktop computer, which performs text and image processing, numerical computations, communication with peripheral devices or resources in the world wide web. Therefor a processor is used that consists of hundreds of millions of transistors integrated into a small silicon chip. The operating system installed on such a computer leads the user to believe that many programs can be run at the very same time, but this so-called multitasking is just an illusion of parallelism. A typical off-the-shelf CPU, mostly being a hybrid of von Neumann and Harvard architectures [12], can perform computation only in a strictly successive way. Since the temporal resolution of human visual perception is much lower than the possible frequency of a desktop computer to switch between the execution of different tasks, the illusion easily can be uphold.

Another device extensively utilized – hopefully the reader sometimes recognizes this – is the human brain. Not only the author's, but a lot of other people's brains, too. These people manage to communicate with each other, while in their heads visual, tactile and auditory stimuli have to be processed rapidly, continuously and in a highly parallel way. Useful information has to be extracted, combined and interpreted in real-time, i.e. within a definite very small time window. Complex sequences of conscious thoughts occur – maybe about what to say next, about the social role within the discussion group and what lunch will be today. Not to mention subconsciousness... The human brain is a *real* multitasking computer.

Until today it remains an unsolved challenge to satisfyingly understand how the computational power, consciousness, will and intellect can arise from the complexity of this neural network. As we know a lot about basic processes in the scope of cellular mechanisms, one way to go on in the search for essential principles is clearly bottom up. This means finding out which of the cellular behaviors are indispensable to preserve the flexibility and computational abilities of such a system and which can be skipped.

The *Electronic Vision(s)* group at the Kirchhoff Institute for Physics in Heidelberg has developed an application specific integrated circuit (ASIC), which implements a network of artificial neurons with binary in- and outputs [26]. The chip is extensively tested and a powerful software interface and detailed documentation have been created. Many different applications for this system have been proposed and performed already [29, 9, 20, 7, 28] while the project is still growing. But the applied network model is far from being biologically realistic, because it neglects basic features of real brain cells. The most important one is the coding of information via temporal patterns of stereotypical signals called *spikes*.

Recent results from neuro-science allow to extend the existing hardware neuron model towards biology while still utilizing the device itself as the core of computation. A basic road-map set up for this thesis was

- to gather the necessary knowledge from neuro- and computer-science
- to create a first concept of a spike interpretation environment for the Perceptron
- to create a flexible software implementation in order to cut down the emerging parameter space
- to implement the developed solutions on the programmable gate array controlling the Perceptron (the tutor's job)
- to gain specifications of the system
- to utilize this novel tool to simulate cortical neural networks

Since all points of this task list have been fulfilled more or less successfully, a third category of computation can be claimed to be used for this thesis: An artificial neural network was put into action to advance it towards a more biologically realistic operation mode and to study its behavior.

Most of the conceptual planning concerning this system has been done in cooperation with the tutor of this work, Michael Reuss. The *Electronic Vision(s)* group enabled and supported the work by generously providing knowledge, tools and ideas. Most of the software necessary for the operation of the novel system was written by the author. Essential fragments programmed by other members of the group that have been integrated into the framework without or with just slight modifications will be explicitly mentioned. All experiments proposed in this document have been performed by the author, although some of them make use of ideas and methods developed by others. In all cases the originators will be referred to.

Reading this thesis

The document is structured into three main parts:

- The introduction of basic facts and concepts used for this work, i.e. things that already were on hand in the beginning
- The description of devices and methods that have been developed in the context of this study
- The description of the most important experiments and results

Technical terms that are introduced for the first time usually are written in italic letters. Once explained, they will be assumed to be known for the following text. In most cases the meaning of abbreviations or acronyms becomes clear the first time they appear, only very common ones are not explained. A complete list of all acronyms and abbreviations is appended at the end of this thesis on page 105. A few comments on the figures: Experimentally measured data points plotted in graphs often are connected by lines. Obviously these lines do neither represent real data nor are they to be understood as a fit. They just serve to facilitate the finding of corresponding data points. Unless otherwise noted, schematics and technical drawings have been created by the author.

Chapter 1

Introducing Facts and Concepts

Biological organisms obtain their computational capabilities from systems of massively interconnected cells. Those basic processing elements, called neurons, build a very dense network in vertebrates' brains (in Latin: *cerebrum*). The *cerebral cortex* is a fragile nervous tissue and forms a part of the brain's surface. It holds most of cerebral neurons and spreads across an area of about $1.5\,\mathrm{m}^2$ due to its wrinkled and folded topology. The neuron density in the human cortex is more than 10^4 neurons per cubic millimeter. Each neuron in the cortex is connected to 10^2 up to 10^4 other neurons, sometimes across very long spatial distances. For centuries scientists of different disciplines have been researching the functionality of the brain, and today they have a detailed knowledge to their disposal.

Very much is understood about processes in the scope of a single neuron cell, but mechanisms determining the behavior of neuron populations are much more difficult to extract. Different approaches have been developed, in particular real life measurements and simulations, but also methods from statistical physics, neuro-psychology and others.

Examples for non-invasive measuring methods of brain activity are electro-encephalograms (EEG) or functional magnetic resonance imaging (fMRI). Imaging with voltage sensitive dyes (VSD) is a relatively new, invasive and extracellular in vivo recording strategy [21], while neuro-science is gaining experience with intra- and extracellular patch clamp techniques already for decades [34], usually applied in vitro.

There is a wide spectrum of different approaches to simulate neurons and neural networks. Nearly all of them skip the effort to exactly describe all known cellular mechanisms in order to make the model computable in reasonable time, allowing for individual demands and individually available resources. Nowadays the standard for modeling neurons with high accuracy is the Hodgkin-Huxley model [3], named after its two originators. It describes many features of neuron dynamics in a very detailed way, but due to its CPU-intensive computation practically only very small numbers of interconnected neurons can be simulated.

Neural coding and network dynamics are often studied utilizing more phenomenologically oriented models with less complexity. The so-called Integrate-and-Fire model (I&F), which will be introduced in sec. 1.2.4, is a powerful representative of this category. It ignores many details of intracellular dynamics, but is sufficient in many cases and applicable to large neuron populations. "Large" means numbers in the order of $\sim 10^2$ to 10^4 cells, depending on the level of sophistication and especially the degree of connectivity and simulated activity.

Most simulations of neurons and neural networks are plain software approaches, only few of them utilize hardware, e.g. FPGAs or full custom designs [17, 25, 19]. This study is on

simulating populations ($\sim 10^2$) of neurons using a VLSI implementation of a simple neuron model. An FPGA controlling this chip and hitherto mainly performing data in- and output administration was used to advance the neuron model on runtime (see sec. 2.1).

1.1 Biological Neurons

A schematic of a single neuron is shown in fig. 1.1, a real pyramidal neuron from a rat's cortex can be seen in fig. 1.2. The size of a mammal's neuronal cell body, or *soma*, ranges from 5 to 100 μm. It has some wire like extensions connecting it to other neurons mostly in its close surrounding area, but sometimes even across centimeters. In terms of information flow some of them are inputs (*dendrites*), receiving signals from other neurons, and one of them is an output (*axon*), conducting the signal away from the soma. Axons fan out into axonic trees and distribute information to several target neurons by coupling to their dendrites via *synapses*.

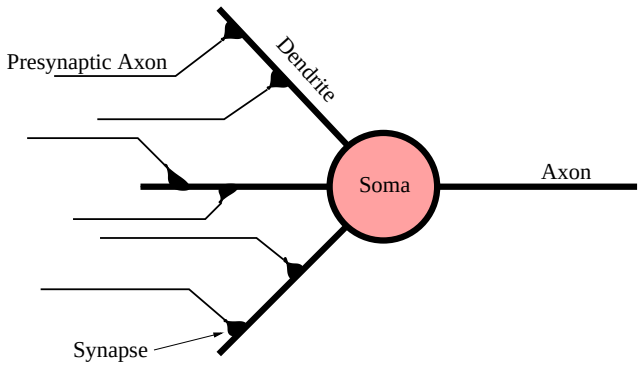


Figure 1.1: Schematic of a single neuron cell. The neuron consists of a cell body (soma), a tree of cell extensions called dendrites and another extension, the axon. The neuron receives information via synapses from axons belonging to other cells. Thus synapses link different neurons. The axon conducts signals generated by the neuron itself away from the soma to other synapses. The way of information coding is explained in the text.

As already mentioned above, a cortical neuron can have up to 10^4 synapses. These interneuronal links modulate arriving information in terms of the effect on the postsynaptic [1] neuron. Thus synapses are said to "weight" the input. The functionality of a neural circuit strongly depends on the composition of these weights. The possibility of manipulating them is called synaptic *plasticity*. Strengthening and weakening synapses is a basic way of learning within the brain.

A neuron's cell membrane divides intracellular from extracellular space. In the absence of any input the electric potential between interior and outer cell space differs. This voltage called *resting potential* has a value of about $-65\,\mathrm{mV}$. It represents the steady state of concurring voltage-dependent ion channels increasing respectively decreasing the *membrane potential*, in neuro-science also called *polarization*. Under the influence of input the membrane potential is hyper- or depolarized. There are two different categories of synapses, depending on the effect of input signals arriving there. If depolarizing the membrane, the synapse is called *excitatory*, otherwise *inhibitory*.

Neural signaling happens via temporally very short positive excursions of the membrane potential, called *action potentials* or *spikes*. Such a spike is triggered by the membrane

[1] postsynaptic = located *after* the synapse in terms of information flow

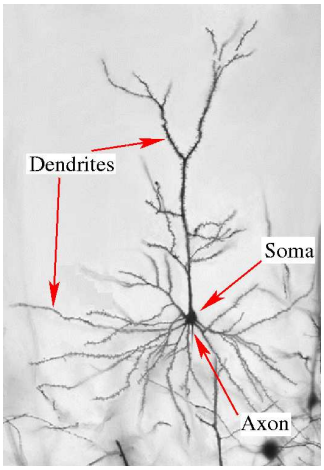


Figure 1.2: Photo of a golgi stained pyramidal neuron from a rat's cortex. The main component parts of the cell are indicated. Microscope magnification is 250x. Courtesy of Grazyna Gorny.

potential exceeding a distinct threshold value and, while its extension is locally bounded, travels along the axon. Since spikes produced by a distinct neuron always look the same, information has to be completely coded by their firing times.

After the occurrence of an action potential and some time of *hyper-polarization* [2] the membrane potential quickly returns to its resting potential (if not dragged away from it again by new input). The minimum time t_{ref} between two subsequent spikes is referred to as a neuron's *absolute refractory period*. After this time a *relative refractoriness* still inhibits the neuron but does not completely prevent it from firing again. This inhibition quickly dies down with a time constant in the same order as the absolute refractoriness.

Reaching a synapse, an action potential initiates some biochemical mechanisms leading to a change of the post-synaptic dendrite's membrane potential (*post-synaptic potential* or PSP). See fig. 1.3 for illustration. Many PSPs add up within the soma and determine the neuron's membrane potential. In scenarios of low to normal spike rates impinging a single synapse even the synapse itself nearly linearly superposes its PSPs, but if pre-synaptic rates get too high linearity breaks down since not enough neuro-transmitters are available anymore.

Postsynaptic potentials decay within time-scales of about $3 - 30\,\text{ms}$, an action potential happens in the scope of 1 ms. Let $v^j(t)$ be the membrane potential of a neuron j and let $\bar{v}$ be its *spiking threshold*. Typically $\bar{v}$ is about $20 - 30\,\text{mV}$ above j's resting potential. Regarding the fact that a single spike arriving at one of j's excitatory synapses triggers a depolarization peaking in the range of only 1 mV many presynaptic [3] spikes within a short time window are necessary to make $v^j(t)$ cross $\bar{v}$. See [18] for more information about neurotransmitter control of cortical activity.

[2] hyper-polarization = membrane potential is even more negative than the resting potential
[3] presynaptic = located *before* the synapse in terms of information flow

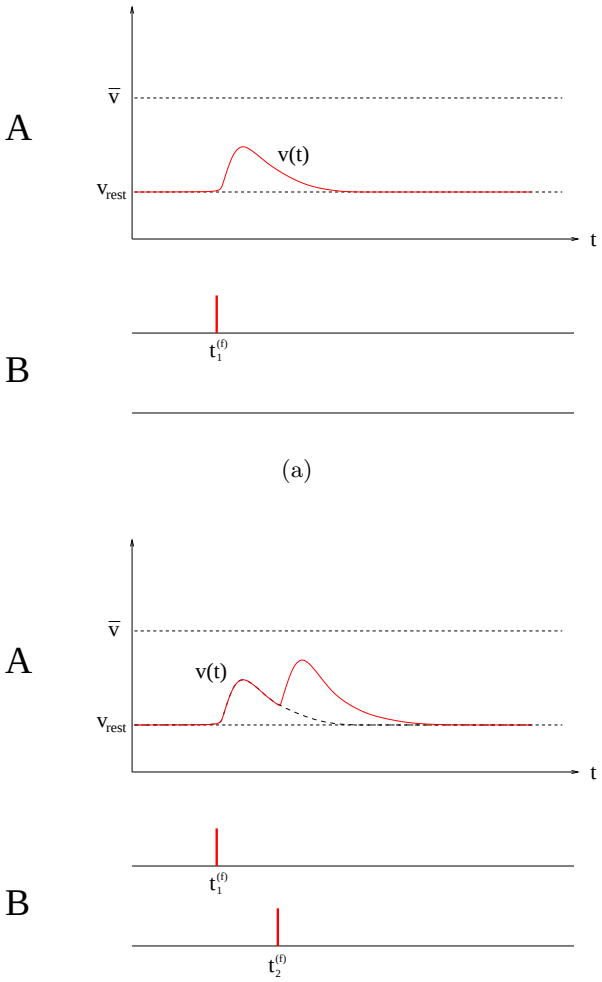


Figure 1.3: Membrane potential $v(t)$ (A) depending on two presynaptic inputs (B). The jth spike arriving at synapse i at spike time $t_i^{(f_j)}$ causes a PSP (a). If $v(t)$ already is above its resting potential a new PSP adds to the course (b). In case many PSPs make $v(t)$ cross the spiking threshold $\bar{v}$ an action potential (which exceeds the scale of the plot) is fired (c). Afterwards $v(t)$ runs through a phase of hyper-polarization.

1.2 Modeling Neural Networks

The following section will give a short review about the main examples of neural network models and their integration into a classification scheme ("generations") suggested by Wolfgang Maass [13].

1.2.1 First and Second Generation Neurons

Fig. 1.4 shows a simple model of a single neuron. It consists of a functional body (blue area), an input vector $\boldsymbol{x}$ connected to this body via a vector of weights $\boldsymbol{w}$ and an output y. The value of y is determined by a function $F(s)$, the argument s being defined by the sum of all weighted inputs $\boldsymbol{x} \cdot \boldsymbol{w}$ plus some threshold $-\delta$.

$$y = F\left(\left(\sum_i x_i w_i\right) - \delta\right) \quad . \tag{1.1}$$

The neuron's inputs, its output and the weights are real numbers, x_i, y, $w_i \in \mathbb{R}$. The bias δ typically is a positive constant. $F(s)$ is called *transfer* or *activation function*, it usually is represented by a strictly monotonically increasing function of s with a bounded co-domain.

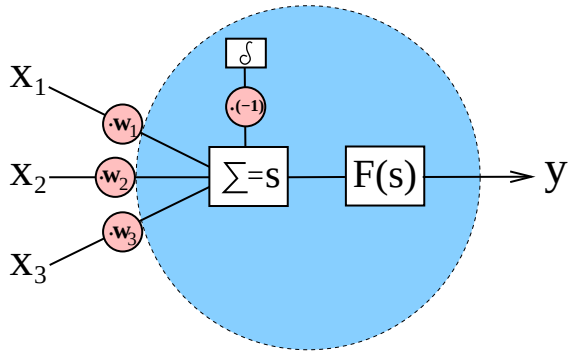


Figure 1.4: Scheme of a simple neuron model. The output y is determined by a function $F(s)$. The argument s is the sum of all weighted inputs minus some threshold, $s = \boldsymbol{x} \cdot w - \delta$.

Obviously the four basic elements of a biological neuron, namely the soma (summation and modulation), the dendrites (inputs), the axon (output) and the synapses (weights) are already contained in the model. The integration of inputs can be motivated by the summation of postsynaptic potentials within real somata (see sec. 1.1). In [13] these kinds of neuron descriptions are classified as *second generation* models. Hitherto the *first generation* was withheld because it easily can be derived from the second generation.

In 1943 the neuro-physiologist Warren McCulloch and the mathematician Walter Pitts suggested a neuron model representing a special case of the one described above. Fig. 1.5 shows a schematic of what is nowadays commonly known as a *McCulloch-Pitts neuron*. The arbitrary activation function in eq. 1.1 is replaced by the unit step function $\Theta(\cdot)$, hence the output is drawn from $\{0, 1\}$. Since the inputs are thought to be other neurons' outputs, they consequently have to be binary, too.

$$y = \Theta\left(\left(\sum_i x_i w_i\right) - \delta\right) \qquad y,\, x_i \in \{0, 1\} \quad . \tag{1.2}$$

Eq. 1.2 describes the neuron model to sum up all inputs weighted with a distinct value and compare the sum with a given threshold value δ. If the so-called *inner state* $s(\boldsymbol{x}) \equiv \boldsymbol{w} \cdot \boldsymbol{x} - \delta$ exceeds zero, then the output is true, else false.

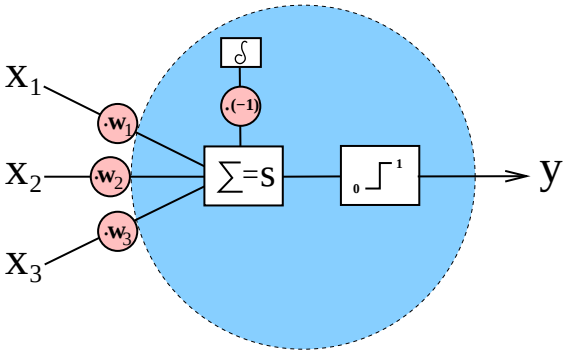


Figure 1.5: Scheme of a McCulloch-Pitts neuron. The binary inputs $x_i \in \{0, 1\}$ are multiplied with their weights w_i and then summed up. If this sum exceeds some distinct threshold $-\delta$, the output is one, else zero.

Given fixed vectors $\boldsymbol{x}$ and $\boldsymbol{w}$, the bias δ in the McCulloch-Pitts model determines whether the output is active or not. This strongly reminds of the spiking threshold in real neurons described in sec. 1.1. Due to the fact that the McCulloch-Pitts model is even simpler than the one defined by eq. 1.1 and because it was the historically first attempt to abstract neuronal dynamics to mathematics, all similar kinds of binary neuron models or "threshold circuits" are classified as *first generation* neurons.

1.2.2 Learning

Configuring synaptic weights in order to make the neuron or neural network perform a given task is usually referred to as *training*. The network itself is said to be *learning* during this process. A basic distinction has to be made between *supervised* and *un-supervised* learning. In the first case a sort of higher authority trains the network from outside. The synaptic weights are manipulated accounting for the discrepancy between a given target function and the network's real output. Supervised learning on the level of synaptic plasticity is not very realistic, since biological neural networks manage to organize themselves. They optimize their capabilities by creating, deleting, strengthening and weakening synapses without any supervision or super-ordinated target function. This is called un-supervised learning, i.e. the network autonomically changes its configuration without having a target function to its disposal.

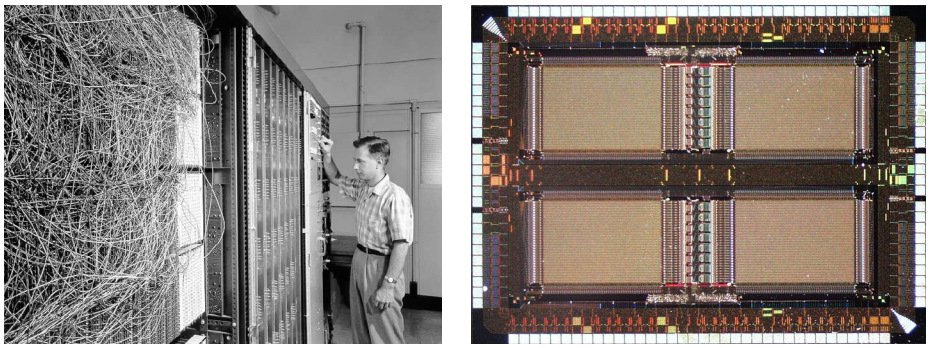
A first important postulation about the mechanisms of self-organization in the brain was made by the psychologist Donald Hebb in 1949, especially notable for it was formulated on purely theoretical grounds. He suggested a dependency of the development of synaptic strengths on temporal correlations between pre- and postsynaptic spikes: "When an axon of cell A is near enough to excite cell B or 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."

Respecting this, today un-supervised learning merely depending on spike time information is called *Hebbian learning*. A very elaborated model of un-supervised Hebb-style learning, the *spike-timing-dependent plasticity* (STDP), is described in [31].

1.2.3 Perceptron

A network consisting of McCulloch-Pitts neurons was presented by the psychologist Frank Rosenblatt in 1958 [23, 24]. His *Perceptron* is the archetypal artificial neural network (ANN) and shows typical features of today's ANNs, including the ability to learn, to generalize, self-organization and fault-tolerance [5].

Perceptrons are commonly trained in a supervised way, often using gradient or evolutionary strategies. Many applications, most of them in the field of pattern recognition, have successfully been developed for this kind of network model. Examples for evolutionary approaches using a VLSI implementation of a Perceptron can be found in [6, 7, 9].



(a) (b)

Figure 1.6: (a) Charles Wightman in the late 1950's in front of Mark I, a Perceptron implementation he had designed. This huge analog electro-mechanical neuro-computer had 512 adjustable synaptic weights, a size in the order of many m^3 and, connected to a 400 pixel camera, could be successfully trained to recognize characters. Picture taken from [5]. (b) A prototype of a Perceptron VLSI implementation designed in the Electronic Vision(s) group in Heidelberg in 2002. This chip (in what follows it will be called HAGEN) has 256 output neurons with 128 adjustable input synapses each, its dimension is about $3 \times 4\,\text{mm}$.

The HAGEN chip - a VLSI Perceptron

In fig. 1.6(b) a VLSI implementation of a Perceptron is shown. The chip (in what follows it will be called HAGEN – Heidelberg AnaloG Evolvable Network) was designed in the *Electronic Vision(s)* group at the Kirchhoff Institute for Physics in Heidelberg and represents the prototype of a larger chip to be built in near future. The prototype already holds 256 McCulloch-Pitts neurons with 128 adjustable input synapses each. The neurons can be configured to form a single- or a multi-layer perceptron with optional feedback. The on-chip feedback wires provide many, but not all possible connections (see sec. 2.1.1).

The chip can be used for multiple types of experiments. It is connected to a PC via programmable logic and discrete electronics. A user-friendly software (see sec. 2.1.2) hides most of the complex chip controlling and provides a comfortable interface to configure synaptic connections and weights. It is possible to feed the artificial neural network (ANN) with input

patterns and to record the output. Due to its digital in- and output (I/O) signals, learning strategies using gradient information cannot be applied. Thus the whole HAGEN system is optimized for the usage of chip-in-the-loop algorithms. For example the functionality of the field-programmable gate array (FPGA) interfacing and controlling the chip can be expanded by an evolutionary co-processor [27] to speed up learning.

The HAGEN chip forms the hardware core for all experiments presented in this study.

1.2.4 Spiking Neuron Models (Third Generation)

Perceptrons are powerful devices [13] and show some basic characteristics of real neurons, but their information processing differs from that found in nature in some basic points: Real neurons do not have binary outputs, information is coded in the spatio-temporal pattern of action potentials. Hence, an event-based model predicting spike-times without exactly modeling membrane mechanisms within each neuron is a promising approach that can be realized for large neuron populations with high connectivity. The Integrate-and-Fire model is such a way to simulate neural networks. It is very popular for studying network dynamics. Maass refers to all kinds of spiking neuron networks exhibiting the possibility of temporal coding as *third generation* models [13].

Leaky Integrate & Fire Model

The *leaky I&F model* [3] describes a neuron's spiking activity using a circuit of simple analog standard devices, see fig. 1.7. For a neuron within a network, $I(t)$ denotes the total current coming from all synapses connected to it. Nonetheless any arbitrary current can be applied, e.g. in order to simulate current injection with an electrode.

If a presynaptic spike runs into a synapse, it will be low-pass filtered and will induce a temporally stretched input current pulse. The spike itself does not have to be described more exactly since it has a stereotyped shape. In the I&F model an action potential is just a formal events characterized just by its firing time t_f. Traveling times along axons may be considered. The synaptic answer to a spike is not an explicit part of the model and can be chosen with arbitrary accuracy regarding knowledge about real synapses.

The current $I(t)$ is split up into two components $I_R(t)$ (passing through the resistor R and responsible for the term *leaky*) and $I_C(t)$ (charging the capacitor C). The voltage $u(t)$ across the capacitance C denotes the membrane potential of the simulated neuron. It is permanently compared to a constant threshold voltage U_{thresh}. If it crosses this voltage from below, a spike will be fired and the firing time t_f is recorded.

The course of the current I_C is given by the change of the capacitor's charge:

$$I_C(t) = \frac{\mathrm{d}q_C(t)}{\mathrm{d}t} = C\,\frac{\mathrm{d}u(t)}{\mathrm{d}t} \quad . \tag{1.3}$$

The current through resistor R is

$$I_R(t) = \frac{u(t)}{R} \quad . \tag{1.4}$$

The total current $I(t)$ therefore is

$$I(t) = \frac{u(t)}{R} + C\,\frac{\mathrm{d}u(t)}{\mathrm{d}t} \quad . \tag{1.5}$$

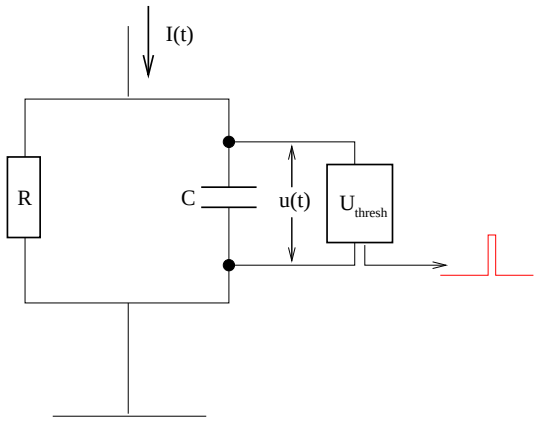


Figure 1.7: Circuit representing the Integrate-and-Fire model of a neuron.

This is a basic differential equation describing the course of $u(t)$ depending on $I(t)$. The standard form is obtained by multiplying eq. 1.5 by R and introducing the time constant $\tau = RC$.

$$\tau \frac{\mathrm{d}u(t)}{\mathrm{d}t} = -u(t) + R\,I(t) \quad . \tag{1.6}$$

The membrane potential $u(t)$ follows eq. 1.6 for all times between two consecutive spikes. Immediately after $u(t)$ has crossed U_{thresh} at time t_f it is set back to a reset voltage $U_{reset} < U_{thresh}$.

$$\lim_{t \to t_f, t > t_f} = U_{reset} \quad . \tag{1.7}$$

After this reset eq. 1.6 determines the course again.

1.2.5 High Conductance States

The general subject of this study is to model the spiking activity of a network consisting of cortical neurons. The inspiration for using a binary Perceptron to simulate neurons with the claim of being biologically realistic was literature about so-called *High Conductance States* (HCS) describing a scenario of high membrane conductances for cortical neurons under awake activity, performing computation. Those states cannot arise for isolated single cells but only for a neuron population (e.g. a cortical layer) resulting from distinct global activity or its simulated presence. In order to describe this phenomenon and to motivate basic assumptions

Membrane model

According to [3], the total current through a cortical neuron's membrane is described by the equation

$$\begin{aligned} I_{tot}(t) &= \frac{\mathrm{d}v_P(t)}{\mathrm{d}t} C \\ &= -g_L[v_P(t) - V_R] - g_{PE}(t)[v_P(t) - V_E] - g_{PI}(t)[v_P(t) - V_I] \quad . \end{aligned} \tag{1.8}$$

C denotes the membrane capacitance, $v_P(t)$ the membrane potential, g_L the leakage conductance, V_R the leakage reversal potential, V_E the excitatory and V_I the inhibitory reversal potential. The subscript $P \in \{E, I\}$ describes the type of the neuron (E = excitatory, I = inhibitory). If a second subscript occurs, it indicates the type of the presynaptic cell.

The time-dependent conductance g_{PE} (g_{PI}) is due to extra-cortical input via the lateral geniculate nucleus (LGN), excitatory (inhibitory) noise coming from other cortical areas and excitatory (inhibitory) activity within the modeled area itself. The components will be defined explicitly later in this section.

Eq. 1.8 is the standard differential equation for a neuron's membrane potential v_P, although it is not an entire description. Additional information is needed about the potential when crossing the spiking threshold $\bar{v}$ (see sec. 1.1).

Values found in [30] are $C = 1\,\mu\mathrm{F\,cm^{-2}}$, $g_L = 50 \cdot 10^{-6}\,\Omega^{-1}\,\mathrm{cm^{-2}}$, $V_R = -70\,\mathrm{mV}$, $V_E = 0\,\mathrm{mV}$, $V_I = -80\,\mathrm{mV}$ and $\bar{v} = -55\,\mathrm{mV}$.

Following [30] eq. 1.8 can be written as

$$\frac{\mathrm{d}v_P^j(t)}{\mathrm{d}t} = -g_L v_P^j(t) - g_{PE}^j(t)[v_P^j(t) - V_E] - g_{PI}^j(t)[v_P^j(t) - V_I] \quad . \tag{1.9}$$

The superscript j is a vector of coordinates and indexes the neuron's spatial location within the network. All potentials have been normalized by the transformation $v \to (v - V_R)/(\bar{v} - V_R)$. In this representation the now dimensionless spiking threshold becomes one, the reset potential zero, $V_E = 14/3$ and $V_I = -2/3$. Furthermore, all conductances are defined as rates with dimension $\mathrm{s^{-1}}$ by dividing eq. 1.8 by the membrane capacitance C. A conductance defined this way transparently indicates the time-scale it represents, i.e. $\tau_g = g^{-1}$.

The time-dependent conductances can be written as

$$\begin{aligned} g_{PE}^j(t) &= F_{PE}(t) + S_{PE} \sum_k a_{j-k} \sum_l G_E(t - t_l^k) \quad , \\ g_{PI}^j(t) &= f_{PI}(t) \ + S_{PI} \sum_k b_{j-k} \sum_l G_I(t - T_l^k) \quad . \end{aligned} \tag{1.10}$$

Again P is in $\{E, I\}$, $F_{PE}(t)$ is defined by $F_{PE}(t) = g_{lgn}^j(t) + f_{PE}^0(t)$ and $t_l^k(T_l^k)$ is the time of the lth spike of the kth excitatory (inhibitory) neuron. Since input from the LGN normally only excites its target neurons, these equations are slightly asymmetrical for excitatory and inhibitory drive. The noise coming from outside the modeled area is represented by the stochastic conductances $f_{PP'}^0(t)$. A method for estimating this background noise is suggested in [30].

Typical time-scales in the human visual cortex are those of extra-cortical input via the LGN, $\tau_{lgn} = O(10 - 10^2\,\mathrm{ms})$, base cellular leakage time-scales, $\tau_L = g_L^{-1} = 20\,\mathrm{ms}$, and time-scales of cortico-cortical interactions (which are of primary interest in this study), $\tau_{syn} = 4\,\mathrm{ms}$ [30].

Useful implications of High Conductance States

The High Conductance State is the normal operating point of an awake human cortex under stimulation. It is characterized by a large total conductance $g_T(t) \equiv g_L + g_E(t) + g_I(t)$. In vitro (no global activity!) and in vivo measurements in neurons show an increase in membrane conductance of factor three to five [2] resulting from the presence of activity. Simulations of

the visual cortex show up to ten-fold increase in conductance from completely un-stimulated to highly stimulated state [30]. According to [2] neurons under normal conditions and activity often have a depolarized average membrane potential (-65 to $-60\,\mathrm{mV}$ instead of -80 to $-70\,\mathrm{mV}$ without activity) and membrane potential fluctuation amplitudes being at least ten times larger compared to those in an un-stimulated scenario. These fluctuations close to the spiking-threshold can cause a significantly higher spontaneous firing rate. At this point the time-scale corresponding to a neuron's membrane total conductance ($\tau_g \simeq 2-4\,\mathrm{ms}$) becomes dominant in neural information processing. This fact has some important consequences and will allow a more convenient neuron model compared to the standard leaky integrator.

For convenience, eq. 1.9 is simplified by dropping the neuron type specifier P and its spatial indices and transforming it to

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = -g_T v(t) + I_D(t) = -g_T(t)[v(t) - V_S(t)] \quad . \tag{1.11}$$

The so-called *difference current* I_D is defined by

$$I_D(t) = I_D[g_E(t), g_I(t)] \equiv g_E(t)V_E - g_I(t)\,|V_I| \quad , \tag{1.12}$$

the *effective reversal potential* V_S by

$$V_S(t) = V_S[g_E(t), g_I(t)] \equiv I_D(t)/g_T(t) \quad . \tag{1.13}$$

It is clear from eq. 1.11 that $v(t)$ follows $V_S(t)$ with a time-constant $\sim g_\tau^{-1}$. This means that in High Conductance State the membrane potential $v(t)$ closely tracks the course of $V_S(t)$, which was quantitatively shown by the authors of [30] by making an asymptotic analysis and a successful comparison to in-vivo measurements.

Eq. 1.11 also makes clear that $V_S(t)$ has to be larger than one if $v(t)$ shall cross the spiking threshold, because $-\frac{\mathrm{d}v(t)}{\mathrm{d}t}$ must be larger than zero at spike time t_{spike} and $v(t_{spike})$ necessarily is one. Regarding the fact that membrane currents of excitatory and inhibitory drive are nearly balanced in cortical operating regime [30],

$$g_E V_E \simeq g_I\,|V_I| \quad , \tag{1.14}$$

and that the ratio of $V_E/V_I \ll 1$ the inhibitory conductance may be assumed to dominate both the excitatory and the leakage conductance. In [30] the accuracy of this assumption is shown in simulations.

A new way of simulation

Combining these insights the essence of the considerations being made above emerges: As $v(t)$ is slaved to $V_S(t)$ and g_T is dominated by g_I one may write

$$v(t) \simeq V_S = \frac{g_E V_E + g_I V_I}{g_T} \simeq \frac{g_E V_E + g_I V_I}{g_I} = V_I + V_E \frac{g_E}{g_I} \quad . \tag{1.15}$$

V_E and V_I are constants, thus the course of $v(t)$ is determined by the ratio of excitatory to inhibitory conductances. Furthermore, for modeling the neuron's spiking activity not the exact ratio $g_E(t)/g_I(t)$ has to be known but only whether this ratio exceeds some distinct

critical value q_{crit}. Since the spiking threshold in this model is one, eq. 1.15 leads to the spiking condition

$$\begin{aligned} V_I + V_E \frac{g_E}{g_I} &\stackrel{!}{=} 1 \\ \Longrightarrow \; q_{\text{crit}} \equiv \left(\frac{g_E}{g_I}\right)_{\text{crit}} &= \frac{1 - V_I}{V_E} = \frac{5}{14} \quad . \end{aligned} \tag{1.16}$$

The classical Perceptron model seems to be predestined for solving this discrimination task.

The HAGEN chip, which implements a Perceptron with McCulloch-Pitts neurons, integrates its weighted inputs by summing up currents in an analog way (see sec. 2.1.1). This is very close to what happens in a neuron's membrane, see eq. 1.8 and 1.12. The instantaneous integration of the currents on the chip is a good approximation, because in High Conductance State time-scales of total membrane conductance become the shortest of all time-scales involved in neural information processing.

For all these reasons it is a justifiable approach to use the HAGEN chip for simulating High Conductance States. Modeling synaptically induced currents and utilizing the binary outputs to decide whether the current sum is large enough to make the neuron fire is a good approximation for reality. To sensibly reproduce the difference current defined in eq. 1.12 on the HAGEN chip the provided binary inputs might be not accurate enough, although the synaptic weights can be configured with a precision of 11 bits. But dynamically changing the weights would take much too long to obtain simulation efficiency for technical reasons. The problem was solved by pooling more than one binary input for multi-bit (integer) synapses and apply conductance courses at these new inputs.

In conclusion, the HAGEN chip can be configured as a simulator of neurons in High Conductance State without the need for generating or inducing high activity. Short membrane time-scales due to high membrane conductances and a simple spiking condition can straightly be translated to intrinsic features of this Perceptron implementation. The new way of simulation is conductance-based since it takes advantage of consequences from High Conductance State. Yet its computation complexity can be compared to that of a Perceptron model. Sec. 2.1 gives detailed information about the configuration and modifications applied to create a spiking environment for HAGEN.

1.3 Liquid Computing

Previously it was claimed that the question of how intellectual and computational power manifests within the brain has to be solved yet. An important step towards understanding the emergence of the brain's capabilities was made by Wolfgang Maass et al. [14] and independently by H.Jaeger [10]. They suggested a computational model different from the common and well known Turing machine. Both developed a mathematical theory describing computation in recurrent dynamical systems without stable states. In what follows, Maass' work and nomenclature will be referenced.

The liquid computing concept provides anytime parallel processing of multi-modal continuous input data. This is in contrast to common computer systems based upon the Turing theorem which perform given computation tasks in a successive but not real-time [4] and not parallel way.

A universal computer

According to Maass' model, a trained linear readout connected to a "sufficiently complex recurrent circuit" [15] of non-linear devices can, at a time $t > s$, extract information about a continuous input stream u fed into and perturbing this medium at a time s. The circuit acts as a high-dimensional and non-linear filter with memory. In former work Maass provided criteria describing the quality of such a complex filter system [14].

A so-called Liquid State Machine (LSM) necessarily consists of both a liquid and a linear readout which has to be trained according to the users demands. It represents a universal computer for if only the liquid is high dimensional and complex enough it can approximate every function solvable by a finite state machine F. The accuracy of approximation depends on the complexity of the liquid. The fact that a linear readout is sufficient makes training easy and robust [15]. Changing the function that has to be predicted only needs a change of the readout, the liquid itself keeps fixed.

Fig. 1.8 shows the formal scheme of a LSM. The machine itself consists of a filter L^M and a readout function f^M, which has no memory and permanently translates the filter output $x^M(t)$ to some target output $y(t)$. Even if f^M is just a linear readout it can be trained to obtain every (even non-linear) target function of the input $u(s)$ with $0 \leq t - s < \tau_{mem}$. Here τ_{mem} denotes a typical time describing the memory capacity of the liquid.

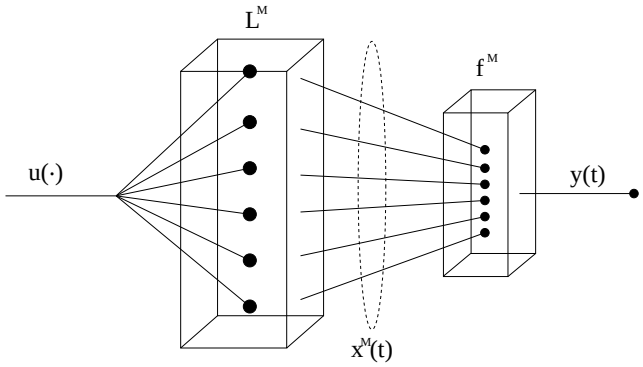


Figure 1.8: Scheme of a Liquid State Machine according to W.Maass. See text for details.

[4] For "real-time computing" the result of a computation is needed within a definite short time-window

A glass of water exposed to some mechanical excitation would fit this model and is an illustrative example of a suitable medium's fluid character. Induced oscillation of the water molecules could be measured via pressure sensors or optical devices and thus be transmitted to some linear readout as plain numbers. In [4] such a real-world implementation of liquid computing is modeled and successfully trained. The capability of spiking neural networks to represent a LSM's filter has been shown by Maass et al. in [15].

1.3.1 Liquid Computing with a Perceptron

In [1] Nils Bertschinger and Thomas Natschlaeger suggested liquids of McCulloch-Pitts neurons with binary outputs taken from $\{-1, 1\}$, working in discrete time domain. They proposed an easy instruction to randomly generate network topologies, given only the following parameters to vary. Be N the total number of neurons within the net. Each of the N neurons then gets k incoming connections from randomly chosen neurons. The weights of the incoming connections are drawn from a zero-centered Gaussian distribution with variance σ^2. Such a network is called *input driven* as the external input signal $u(t)$ is connected to every neuron. For every time step it is drawn randomly from $\{\bar{u} - 1, \bar{u} + 1\}$ with equal chances, $\bar{u}$ being a constant bias.

1.3.2 The "Edge of Chaos"

Bertschinger et al. supposed the computational performance of a liquid represented by a neural network generated according to their instructions to be restricted to a limited region of parameters. They found criteria for their networks to deliver reproducibly well working liquids. Shortly, the two-dimensional parameter space spanned by k and σ^2 (the other parameters being fixed) hosts two characteristic regions of network activity. The first exhibits deterministic behavior, i.e. nearly all neurons within the network will follow the input. The second extreme is chaotic behavior (the term will be justified within this study), as due to strong network connectivity the vector of all neurons' outputs, the "liquid state", shows high activity that seems to be mostly independent of the recent input. Indeed the information saved within this liquid's states very quickly gets lost in both characteristic regions, see sec. 3.2 for a method measuring this loss. A computational and memory performance (see definition of the measure memory capacity below) significantly larger than zero can only be obtained when creating a network with parameters taken from the band separating those two phases in parameter space. The first to describe this border was C.G.Langton [11], he called it the "edge of chaos".

The author of [28] managed to transfer the software simulation experiments of Bertschinger et al. to the HANNEE and HAGEN system and thus was able to let the Perceptron hardware work as a liquid. This also meant to use IO values $\{0, 1\}$ for the inner neurons instead of $\{-1, 1\}$, as Bertschinger did. Hence some modifications regarding the input forcing had to be made, which affected the liquid's performance (see [28] for an analysis of this problem). Another limitation given by the hardware approach was the fact that synaptic weights on the HAGEN chip can not be chosen freely but only from a bounded interval of values. The characteristic edge in parameter space could be shown on this VLSI platform and a measure for the chaotic character of distinct parameter regions described above was developed.

During the work for this study those methods were adapted to the special requirements of a multi-bit and temporally pseudo-continuous approach and have been successfully integrated

into HASTE. Hence it is now possible to perform liquid computing on a VLSI Perceptron with nearly arbitrarily resolved, modeled input shaping and the possibility of spatial and temporal feedback path manipulation. Sec. 2.1 gives more information about technical details.

1.3.3 Memory Capacity

Researching computational performance of a liquid state machine requires a measure, which is, according to Bertschinger's terminology, given by its *memory capacity* (MC). More precisely it values the capability of the system to extract distinct information about the input applied to the liquid some time ago. The larger this time gap may become without prediction of target values significantly worsening, the larger a liquid state machine's memory capacity is said to be.

A liquid state machine M's quality of predicting a distinct target function can be measured via the so-called *mutual information* (MI). For the readout of M and a given input stream a target value $y(t)$ is defined for every time t. The value actually predicted by M is denoted by $v_M(t)$. The mutual information then is

$$\mathrm{MI}(v_M, y) = \sum_{v_M{}'} \sum_{y'} p(v_M{}', y') \log_2 \frac{p(v_M{}', y')}{p(v_M{}')p(y')} \quad , \tag{1.17}$$

where $p(v_M{}') = Pr\{v_M(t) = v_M{}'\}$ and $p(v_M{}', y')$ denotes the joint probability. The setup described here always implies $v \in \{0, 1\}$.

Let the target value $y(t)$ be determined by the input of a finite time window in the past. This window shall have a constant size and the interval between the last time step within this window and t shall be constantly τ. In other words, the time window defining the target output y_τ moves with t, but shifted by τ. The memory capacity of M is evaluated by training it to predict y_τ correctly for each value of $\tau \geq 0$ and sum up the resulting values $\mathrm{MI}(v_{M,\tau}, y_\tau)$.

$$\mathrm{MC}_M = \sum_{\tau \geq 0} \mathrm{MI}(v_{M,\tau}, y_\tau) \quad . \tag{1.18}$$

The example to follow shall illustrate this definition: A liquid state machine with binary IO and perturbed with one single bit-stream is trained to tell the parity of n consecutive input values fed in $\tau, (\tau + 1), ..., (\tau + n - 1)$ cycles ago. Thus, τ denotes some time shift into the past. The problem becomes more difficult to solve the larger τ is, as the liquid has to keep the information about the bits of interest for up to $\tau + n - 1$ cycles.

Per definition, the memory capacity of a liquid is the discrete integral of $\mathrm{MI}(\tau)$ over all values of $\tau \geq 0$. As the liquid state machines regarded here are working in discrete time domain τ is given as a dimensionless integer denoting *number of cycles*, but actually represents a measure for time. The memory capacity can be understood as a time constant of information decay or "echo" within a liquid.

Chapter 2

Architectural Overview & Implementation

In the previous chapter many concepts have been introduced, intending to make the following sections comprehensible. Since the theoretical background and motivation are now given, in this very chapter the more tangible part of the study's fruit will be presented.

Basis for all experiments described in the next chapter was the development of a flexible experimental platform which will be specified below. The design work is considered to be a main element of this study and happened in cooperation with its tutor, Michael Reuss. Roughly speaking, the developed platform extends devices and methods which interface and control a hardware Perceptron in order to simulate cortical neurons in High Conductance States. The system as a whole is new, most of its components are not. Nonetheless all parts involved into the setup have to be specified more or less elaborately depending on the necessity of detailed information for understanding the functionality.

Two main pillars of the simulation platform will be described extensively. On the one hand details of the hardware core will be given, namely the HAGEN chip already sketchily introduced in sec. 1.2.3. This happens in terms of its control, interfaces, features and anomalies. On the other hand the software framework will be introduced, which represents the basis for nearly all software work done during this study.

Then the extensions developed for the programmable logic device in cooperation with and realized by Michael Reuss are presented. Preceding to this hardware implementation a pure software version of the planned system has been created by the author in order to support the design process. This will be described together with software developed to interface and utilize the hardware extensions.

Finally the solutions for special requirements will be proposed, i.e. how to configure the system for distinct purposes.

2.1 The HAGEN Spike Translation Environment (HASTE)

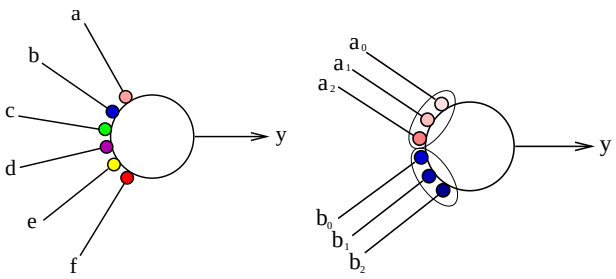
In the introduction the basic idea of using a hardware Perceptron for cortex simulation already was proposed. It was shown that the convenient spiking condition for neurons in the High Conductance State can be simulated utilizing intrinsic features of the VLSI Perceptron HAGEN (see sec. 1.2.5). A key element of such a simulation is the on-chip superposition of conductance courses (CCs) arriving from different synapses.

Since Perceptrons have only binary in- and outputs, there is no way to model a conductance course at a single synapse under normal conditions. Hence, a new idea was to pool a distinct number of binary synapses to one virtual integer synapse, see figure 2.1(a) and (b). By this means, multi-bit values can be applied to the bundled inputs and exceed plain on-off information. But shaping synaptic input into a Perceptron in both time and amplitude has to be performed by some kind of device. Reasonably this device is located in front of the pooled binary synapses, receives an input itself and then generates and applies some resulting information to the multi-bit unit.

In order to generate input courses over more than one cycle this device needs memory. Since the information that has to be transferred to the neuron is thought to represent synaptic conductance courses, such an input generator will be called *conductance course generator*. Fig. 2.1(c) shows the schematic of a first neuron prototype. It receives binary input bit streams at its synapses A and B. These streams are interpreted as *pseudo spike trains*, which has to be explained. The whole Perceptron setup works in discrete time domain, therefore the states of neurons and synapses can only change from one network cycle to the next. During one cycle period T_{sys} a bit can merely exhibit 1 or 0. A pseudo spike train is a bit stream where the value 1 denotes "a single spike occurs during this period" and 0 means "no spike occurs during this period". To simulate real spike trains with this method, the simulated time per network cycle T_{sim} has to be selected. The ratio $\nu_{sim} = T_{sys}/T_{sim}$ is a measure for simulation speed. For $\nu_{sim} = 1$ the simulation runs exactly as fast as the reality it emulates. The presented way of bit stream interpretation has some intrinsic constraints. The maximum codable spike rate is given by the frequency of network cycles $f_{max} = (T_{sim})^{-1}$. The occurrence of a spike cannot be identified with an error less than $\delta_{spike} = T_{sim}/2$. Thus a so-called *interspike interval* (ISI), denoting the period between two subsequent action potentials, cannot be given with an accuracy better than $\delta_{ISI} = T_{sim}$. Hence the simulated time per network cycle T_{sim} should be selected large to keep errors low, but not too large in order to uphold simulation speed. In what follows the inverse of T_{sim}, i.e. the number of network cycles per simulated time unit, will be called *temporal resolution* or ρ_{temp}.

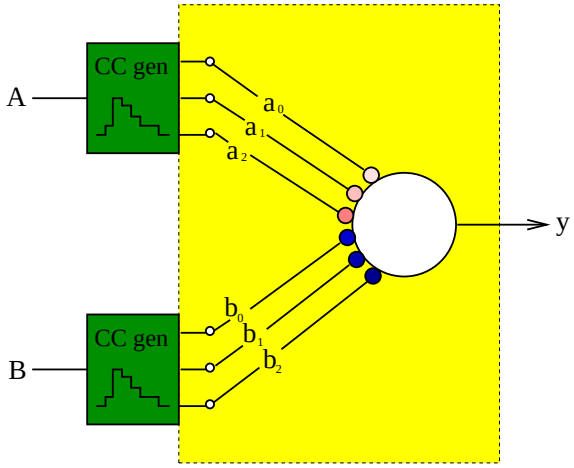
On the one hand, the conductance course generators were developed by Michael Reuss and the author according to their demands and therefore could be designed with nearly arbitrary functionality, including the ability to interprete pseudo spike trains. On the other hand, a hard-wired McCulloch-Pitts neuron cannot be manipulated to *deliver* pseudo spike trains with selectable temporal resolution or even exhibit features like a refractory mechanism. Therefore its the environment of all unmodifiable devices that has to be designed in a way that the binary output can be reasonably interpreted as information about the neuron's spiking behavior. The basic idea to solve this task was introduced in sec. 1.2.5: The output y of a hardware neuron just tells whether the ratio of conductances fed into it is large enough to make the virtual effective reversal potential exceed the spiking threshold (then $y = 1$) or not ($y = 0$). See fig. 2.2(a) for illustration. Consequently, for all times when y is one the neuron is said to fire at its maximum firing rate, for all other times it is interpreted to be quiet.

Since the input of the prototype developed so far has to receive pseudo spike trains, but its output does not deliver these, another processing step has to be implemented between the output of a neuron and another neuron's input. This process must implement a kind of refractoriness by truncating a 1-sequence for a certain dead time. Fig. 2.2(b) shows a scheme of this concept. The output of a neuron including both a refractory mechanism and a conductance course generator at each of its synapses (indicated in light blue in the figure) can be fed back to every other neuron of the same type.



(a) Plain McCulloch-Pitts neuron

(b) Pooling synapses



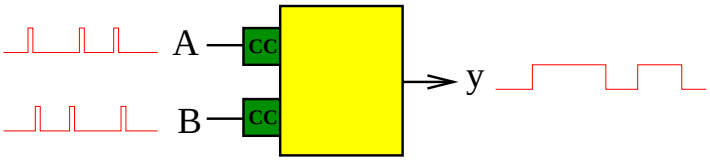
(c) Applying input generators

Figure 2.1: In (a) a single Perceptron neuron with i input synapses (here $i = 6$) is shown. All inputs and the output are binary, the synaptic weights (colored) are drawn from $\mathbb{R}$. As a first step towards a HASTE neuron n synapses (here $n = 3$) are bundled (b). A generator can be connected to every synapse pool, supplying the n binary synapses with binary coded values. Therefore the weights have to be graded, see text for details. In (c) the first prototype of a HASTE neuron can be seen, now having only i/n binary inputs left. The *conductance course generators* interpret their input as being spike-coded, i.e. every cycle the input is 1 denotes that a spike arrives within this cycle's period. Reactively the generators apply signals to the neuron that can take more than one cycle and that can be shaped in terms of their amplitude.

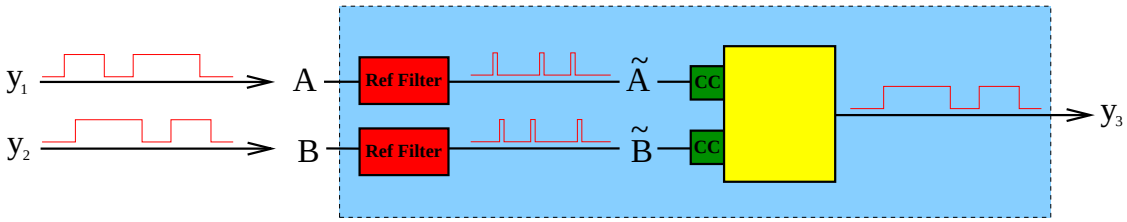
In what follows, the implementation of the neuron model presented so far will be shown. Therefore it is necessary to firstly give more details about the Perceptron chip HAGEN, which was on hand but inalterable, and about its environment, which was manipulated to obtain the desired spike interpretation.

2.1.1 The HAGEN Chip - Technical Details

The *Electronic Vision(s)* group in Heidelberg has developed a hardware Perceptron which implements 256 McCulloch-Pitts neurons with 128 binary inputs each. The resulting 32768 synapses can be given individual weights ranging from -1.0 *HAGEN weight units* (hwu) to 1.0 hwu with a nominal precision of 10 bits plus sign [26]. Each synapse is a current memory cell, it stores a current which is proportional to the configured weight. If this current is positive, it is said to be excitatory, else inhibitory. Each neuron n receives the currents stored by its input synapses, parted into the sum of excitatory components I^n_+ and the sum of inhibitory components I^n_-. The neuron's output o^n basically is determined by the ratio of



(a) Neuron with conductance course (CC) generators



(b) Preprocessing the inputs (y_1 and y_2): refractoriness plus shaping

Figure 2.2: The output of a neuron prototype suggested in fig. 2.1(c) just tells whether the simulated ratio of conductances is above a predefined value (output is 1) or not (output is 0). Thus this output in not spike-coded and therefore cannot directly be fed into another HASTE neuron as it was developed so far, since these prototypes expect pseudo spike trains (a). In (b) this problem is solved by installing so-called *refractory filters*. These filters receive the output of other HASTE neurons (e.g. y_1) and implement a dead time after every cycle this output exhibits 1. For the time being this dead time is a fixed number of cycles and therefore represents an absolute refractory mechanism.

these two currents,

$$o^n = \begin{cases} 1\,, & \text{if} \quad \left|I_+^n / I_-^n\right| > 1 \\ 0\,, & \text{else} \end{cases} \quad . \tag{2.1}$$

Due to the digital I/O of all neurons and the digital weight configuration but the analog summation of synaptic currents this artificial neural network (ANN) is said to be a *mixed mode* device. The analog current integration makes output computation very fast, the digital interface allows high-speed inter-neuron communication and fast interfacing from outside. The change of neuron states is synchronized. If the weights are kept fixed a sequence of input patterns stored in the RAM can be applied with a frequency of 50 MHz. This high speed evaluation of up to $1.64 \cdot 10^{12}$ connections/sec points up a main advantage of the hardware approach. Changing the synaptic weights costs much more time compared to the efficacy of the common operation mode. A maximum weight update rate of $400 \cdot 10^6$ weights/sec can be achieved.

The 256 neurons are divided into four network blocks on the chip, each of them containing 64 neurons. Within each block arbitrary feedback from every neuron to every neuron is possible. Feedback going from one network block to another is not arbitrarily possible due to limited preset wires. Another constraint is the discrepancy of 128 input synapses per neuron versus 256 neurons within the net. Using feedback does not slow down the system.

Fig. A.1 shows the feedback connections that are possible on HAGEN and also illustrates the arrangement of the neurons into four network blocks.

The chip was designed for a fabrication process that is capable to generate spatial structures of down to $0.35\,\mu$m. Therefore a very high synapse density of about $4 \cdot 10^3\,\mathrm{mm}^{-2}$ is achieved. Process variations during chip fabrication have to be compensated by the application of calibration shifts on the synaptic weights in order to uphold synaptic accuracy [26]. Periodic weight refreshing allows for parasitic effects like leakage currents. Due to the density and pure mass of transistors on this chip it is also referred to as a *very large scale integration* (VLSI) design.

The HAGEN chip is controlled by an FPGA via so-called *low voltage differential signals* (LVDS), see fig. 2.3. The FPGA applies the input patterns from a RAM and writes the ANN's output back to the same storage device. It also provides the configuration of the synaptic weights by sending them to digital-to-analog converters (DACs), which prepare the primarily binary values for the analog current memory cells. The programmable gate array itself is accessed by a PC via PCI. The PC runs the administration software (see sec. 2.1.2) and controls the network processing from a higher level.

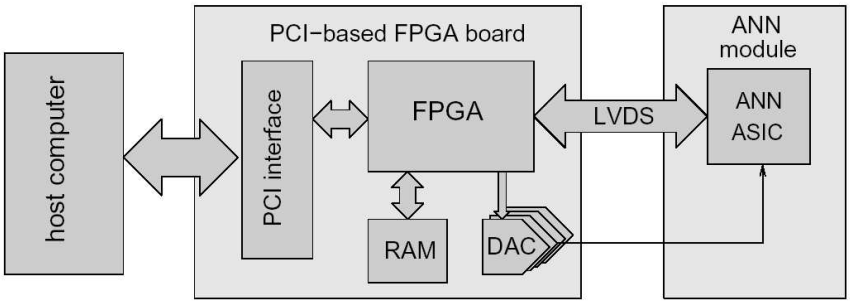


Figure 2.3: The basic architectural setup embedding the VLSI Perceptron HAGEN. The ANN chip is placed on a module which is connected to a PCI-based FPGA board via LVDS. The FPGA provides all control signals and input data for the chip, controls the DAC and stores HAGEN's output. The board itself is connected to a PC running a user-friendly software interface.

2.1.2 The Software Framework

The Electronic Vision(s) group has developed a powerful software framework for controlling and utilizing custom design VLSI devices implementing artificial neural networks. The software is called HANNEE, which abbreviates "Heidelberg analog neural network evolution environment". As the name already tells, a main direction of research on ANNs in this group are evolutionary algorithms for network optimization.

All HANNEE code is written in the languages C and C++. Although the integration of nearly arbitrary network devices is possible, the focus of this section will be on interfacing the HAGEN system, because it was utilized for this study. The HANNEE framework is object-oriented, i.e. in particular the ANN devices themselves are represented by their own classes, including many methods. The software provides a very comfortable graphical user

interface (GUI) allowing to run experiments on the HAGEN system without knowing very hardware specific details. Many applications and tools are already integrated into the project, for example training algorithms for the HAGEN chip and methods for input pattern generation. Compensation of the so-called *fixed pattern noise*, i.e. weight distortions due to process variations during the chip fabrication, is already set up and can easily be applied.

The most important feature of HANNEE in the context of this study is the possibility to integrate new software modules easily. All classes representing real objects or implementing solutions that are applied to such are the last link in a chain of inheritance with very arbitrary classes on the top level. Thus, creating classes similar to already existing objects does not need a completely new description but just a new inheritance of a superior class which describes the new object's functionality and attributes arbitrarily enough. Fig. 2.4 shows an inheritance diagram for the *HagenData* class. A *HagenData* object is the software representation for one network block on the HAGEN chip and holds all information concerning synaptic weights, in- and output data, usage of feedback connections et cetera. The tree exemplarily shows the inheritance principle which was applied consistently throughout the whole HANNEE framework.

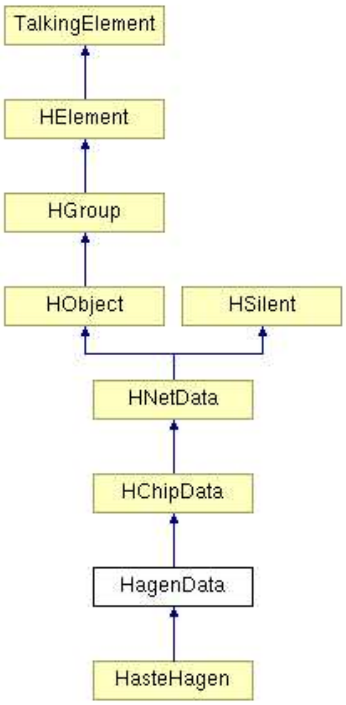


Figure 2.4: Inheritance diagram for the *HagenData* class, the software representation of one HAGEN network block in HANNEE. *HagenData* is derived from *HChipData*, which provides hardware specific interfaces and methods for arbitrary ANN network blocks. *HNetData* interfaces *HChipData* for all non-hardware related classes in HANNEE.

All software that has been written for the implementation of HASTE was integrated into HANNEE. This provides a comfortable usage that is consistent with the appearance of the GUI hitherto. An instruction of how to integrate the HANNEE code extensions into the framework in order to use HASTE is given in sec. A.2.2.

Fig. 2.5 shows a screen-shot of the HANNEE GUI.

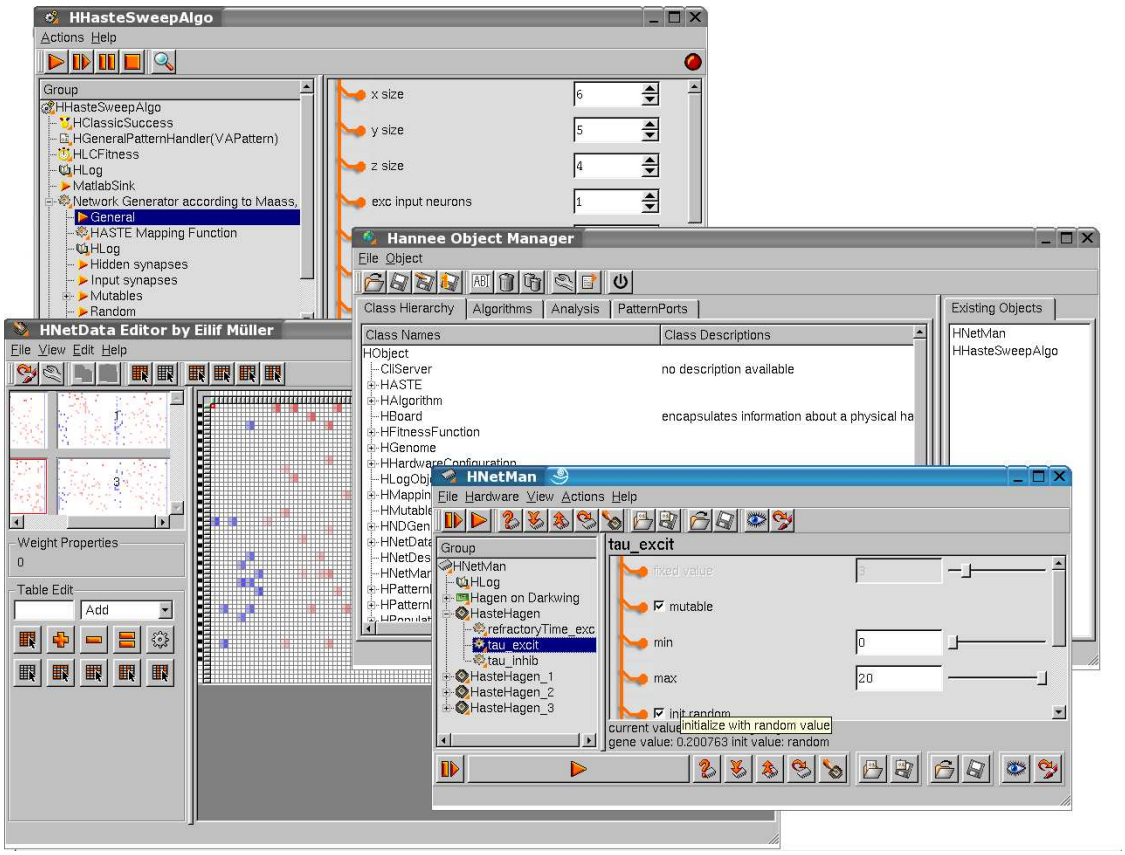


Figure 2.5: A screen-shot of a typical HANNEE workspace. The "network manager" *HNetMan* encapsulates the access to network hardware for the rest of the program. In this picture the *HNetMan* holds four instances of the class *HasteHagen*, which will be introduced later in the text. The *HNetData Editor* is an editor for the weight matrix of all neural network classes derived from *HNetData*, in particular for the *HasteHagen* object and thus for the HAGEN chip configuration, see sec. 2.1.3. The *Hannee ObjectManager* can instantiate and delete all kind of objects available in the HANNEE class tree. The *HHasteSweepAlgo* is a algorithm object including many sub-classes. Among others, a "Network Generator according to Maass" can be seen. This generator will be described explicitly in sec. 2.2.1.

2.1.3 Extending the Components

This subsection describes how the extensions suggested in sec. 2.1 have been transferred to the HAGEN system. The first part will explain the implementation of additional functionality in the controller FPGA. The second parts deals with the new software written for HASTE.

Adding functionality to the FPGA

The FPGA programming was extended in order to realize the refractory filters and the conductance course generators. The VHDL (a hardware describing language) code determining this additional functionality of the reconfigurable device was written by Michael Reuss. In addition to the filters and generators a flexible feedback router was implemented. Since for the operation of HASTE all neuronal signals have to pass through the FPGA (i.e. through the filters and generators) once every network cycle, the implementation of an additional routing device does not slow down the system significantly. The router provides the possibility for all output neurons to be fed back to every other neuron on the chip, independently of the network block the source or target is located on. This new device originally was developed for HASTE but can be utilized independently of it by all HANNEE users. Fig. 2.6 illustrates the principle of routing. The router can delay every signal fed back to the chip. This *global transmission delay* t_{FB} of feedback can only be configured with the same value for all connections and is limited to a region of 0 to 3 network cycles due to the lack of enough programmable hardware.

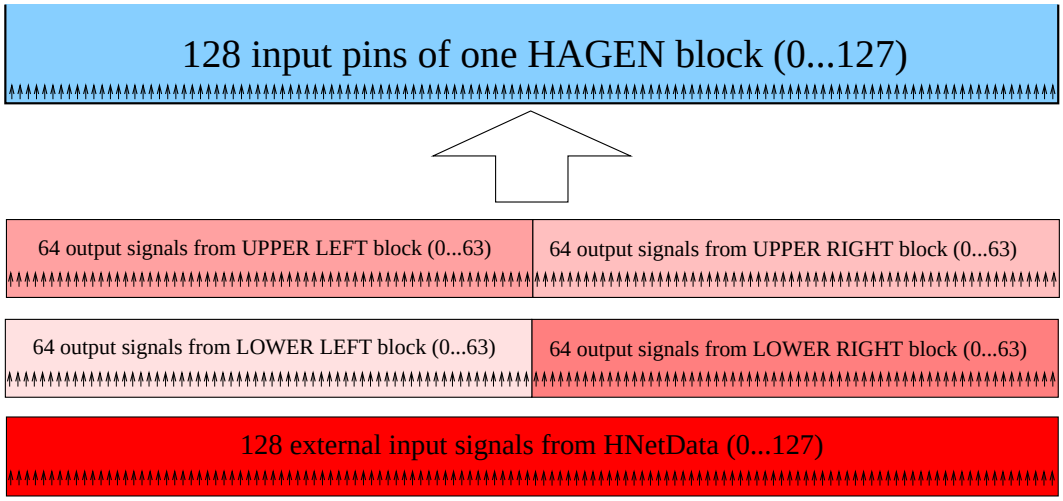


Figure 2.6: The feedback router developed for HASTE has to distribute 128 available input pins per HAGEN network block to 384 possible candidates. Every block receives the output signals from all 256 neurons within the net, divided into four times 64 signals from each block. Additionally 128 external bits provided by HANNEE (more precisely by the *HNetData* object representing this HAGEN block) have to be regarded. These inputs form three rows, i.e. every pin has three possible candidates. The router can select an arbitrary row for every single input pin. Each feedback signal can be delayed by the router for up to three network cycles.

Not all of the 128 available input pins per HAGEN network block were bundled to multi-bit inputs. For the following it is helpful to agree on a block and pin enumeration: The network blocks will be distinguished from each other by their positions in fig. 1.6(b), i.e. upper left, upper right, lower left and lower right, see fig. 2.7. Using the same figure, a pin enumeration will be used going from 0 to 127 for every block, starting at the leftmost pin.

Due to the necessity of calibration (see sec. 2.1.1) two input pins per block were needed to apply constant bias inputs [26], namely the numbers 125 and 126. Additionally, some pins might be required to feed signals directly into the chip, i.e. without being modulated by a refractory filter and a CC generator. This makes sense if, for example, some statistical background has to be simulated which cannot be realized by the output of CC generators. Another reason for not using all available input pins for the application of CCs is the fact that the synapses located next to the edge of the chip (number 127 on both network blocks on the right) respond differently to input than all others. This can be explained with the different surrounding on the chip: The synapses close to the edge are significantly more affected by parasitic capacitances than synapses more distant to the edge.

Hence, eight pins per network block were reserved for direct input without CC generators or refractory filters. For the reasons mentioned above three of them are the numbers 125, 126 and 127. Providing eight direct-ins automatically means that eight of the 128 possible feedback connections cannot be used. Therefore the remaining five were located such that on every block the same output neurons are affected: The pins 124, 60, 61, 62 and 63 are freely available for generating background noise, thresholds et cetera. Providing eight pins for direct-ins means leaving 120 pins for CC generators. This number can be divided by 1, 2, 3, 4, 5, 6, 8, 10, 12 and more. Thus a large spectrum of conductance course resolutions can be realized for these remaining pins without skipping any.

The figures 2.7, 2.8 and 2.9 illustrate the solution that has been developed to implement the refractory filters, the conductance course generators, the direct-ins and the feedback router for the HAGEN chip. They show the HASTE system exemplarily for 4-bit conductance courses and fully characterize the extensions.

Actually all 256 neurons can always be used as output neurons, but not all can be fed back. The direct-ins allow up to 240 feedback connections to CC generators. But if ρ_{amp} bits are used per generator, the number of HASTE neurons that can arbitrarily be fed back decreases to

$$N(\rho_{\text{amp}}) = 240/\rho_{\text{amp}} \quad . \tag{2.2}$$

If for example ρ_{amp} is two, every second output neuron is simply ignored.

The number i_{shape} of shaped inputs per neuron depends on the shape resolution of the generators either,

$$i_{\text{shape}}(\rho_{\text{amp}}) = 120/\rho_{\text{amp}} \quad . \tag{2.3}$$

The number of direct-ins remains unaffected by ρ_{amp}.

The CC generators were implemented as look-up tables, i.e. every time a spike arrives at their input, they start applying a sequence of bit vectors stored in their own registers. These registers have to be filled with reasonable bit arrays provided by the software. Since the FPGA is limited in its size, the registers have a length of 16 bits, i.e. conductance course sequences cannot be longer than 16 network cycles for the time being. This limits the possible temporal resolution. For $\rho_{\text{temp}} = 2000\,\text{steps/sec}$ the maximum length of a conductance course would be only 8 ms, for example.

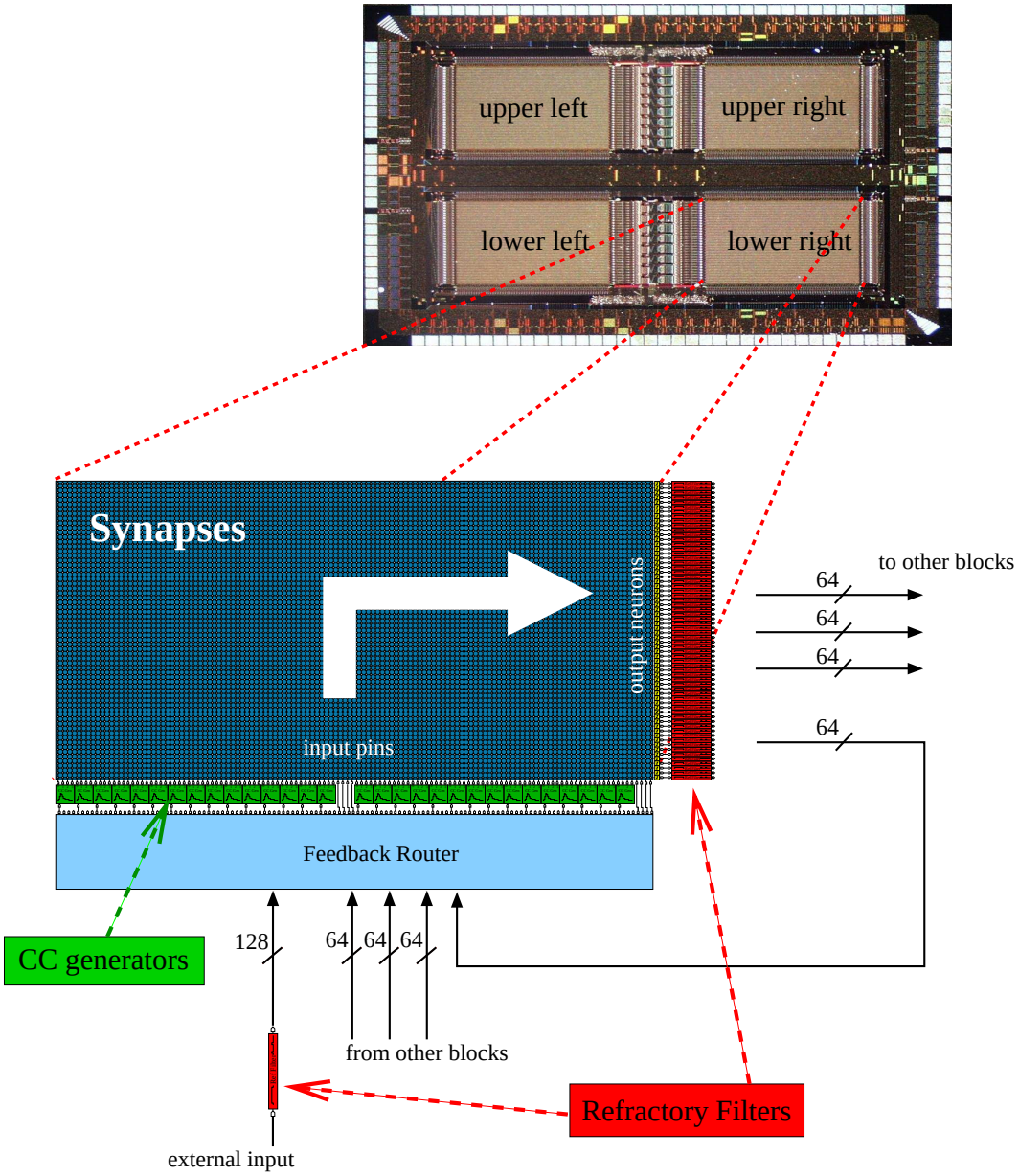


Figure 2.7: Upper: Photograph of the HAGEN chip, including the labeling of the four network blocks. Lower: Schematic of one single HAGEN block (any of the four) with extensions for spike interpretation. The large blue area denotes the synapses. Input data comes from below, the output neurons are located on the right (yellow). Every dot within this area is a possible connection between the input pin under and the output neuron right of it. Every neuron output passes through a refractory filter and is routed to all four network blocks. Consequently the output of all four network blocks arrives at the input. Additionally to these feedback signals, the external input provided by the software competes for the existing pins. The feedback router distributes the candidates to the available CC generators and direct-ins, see fig. 2.6.

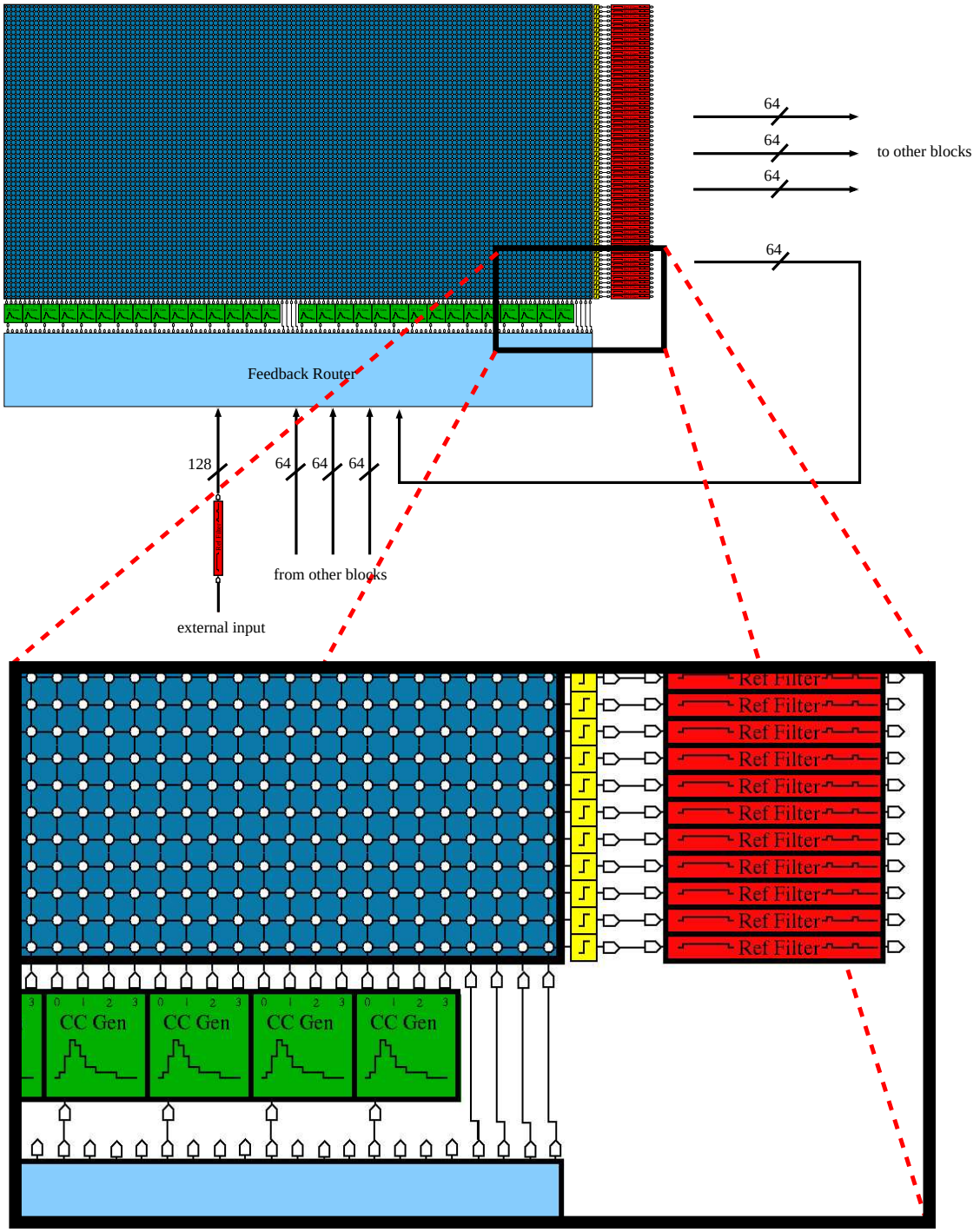


Figure 2.8: Close-up view of a region within the schematic already described in fig. 2.7. Here the synapses, the individual refractory filters, the conductance course generators and four direct-ins can be seen.

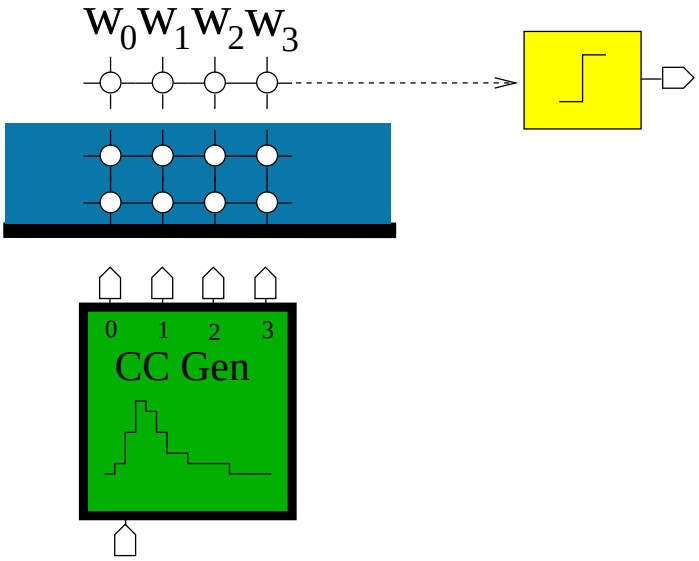


Figure 2.9: Close-up view of a 4-bit conductance course generator with its connection to the synapse columns of a HAGEN block. In this example the generator has four outputs, enumerated from 0 to 3. It applies binary coded integer values (ranging goes from 0 to 15) to the four input pins. This only makes sense if the configuration of the weights w_i allows for the input coding. For every pool of weights the condition $w_i = 2 \cdot w_{i-1}$ must be fulfilled. The effective total weight of the multi-bit synapse is $W_{\text{eff}} = \sum_i w_i$.

Also due to resource limitations, multiple CC generators have to share one look-up table. The maximum number of tables $t_{\text{look-up}}$ is again given by ρ_{amp},

$$t_{\text{look-up}}(\rho_{\text{amp}}) = 30/\rho_{\text{amp}} \quad . \tag{2.4}$$

These $t_{\text{look-up}}$ look-up tables have to be distributed to the $N(\rho_{\text{amp}})$ generators. Let m_{gen} be an enumeration of all generators located on one block, starting and ending at the two outmost ones. In a first step all neighboring generators share one table. In a second step all generators with the same value $\tilde{m} = m_{\text{gen}} \text{mod}(60/\rho_{\text{amp}})$ again use the same storage space. This makes a total sharing rate of eight generators per look-up table. The contents of the CC generators respectively the look-up tables is generated by the software. A complex mapping algorithm has to guarantee that all synapses with different conductance courses are mapped to generators with different look-up tables.

In case the global transmission delay t_{FB} is set to zero the extensions applied to the FPGA increase the duration for one network cycle by 60% compared to the normal HAGEN operation speed. For all values of $t_{\text{FB}} > 0$ there is even no slow-down of the system.

Adding functionality to HANNEE

In order to use the extensions made on the FPGA, the HAGEN administration software had to be adapted and extended as well. A software representation for the new ANN was developed, called *HasteHagen*. This class inherits *HagenData* and therefore can be used by all algorithms that have been designed to interface the HAGEN chip. This is an important

feature, because many methods that have been developed by other members of the *Electronic Vision(s)* group were utilized for the HASTE experiments to be described in the next chapter.

HasteHagen itself includes other new classes that have been designed especially for HASTE, namely multiple instances of the class for CC generators (due to historical reasons it is called *HPSPGenerator*) and one for the extended feedback (*FlexConnectInfo*). For the configuration of the new feedback possibilities a new GUI tab was created for *HChipData objects*. Thus the feedback router can be configured by every user in a convenient way. A *HPSPGenerator* object represents a whole CC generator and can even replace the FPGA generator extension by pure software functionality. This feature was used when the FPGA version of HASTE was not yet ready to work (see sec. 3.1). Using a CC generator purely in software needs communication between HAGEN and the PC every single network cycle. Since this has to happen via the PCI bus, the software approach is very slow compared to the FPGA based operation mode.

More classes encapsulate the discretization of continuous conductance courses to the available amplitude integer values and time steps and the transfer of these courses to the look-up tables on the FPGA. Their functionality and optimal configuration will be explicitly discussed in sec. 3.1.1.

The global refractory mechanism implemented on the FPGA does not have its own software representation, since there is only one single parameter to adjust for the time being. But additionally to the optional software operation mode of the CC generators, a class has been created that provides individually adjustable refractoriness for every single HASTE neuron – again only software-based and therefore slowing down the system significantly.

A pure software operation mode can be selected that emulates the CC generators with more flexibility than the hardware solution It also provides individual refractoriness for every single neuron and a software-based feedback router. In what follows this operation mode will be referred to as *SoftHASTE*.

All new software written for HASTE is well documented, mainly with *Doxygen*[1].

Parameters and graphical user interface

The graphical user interface (GUI) provided by HANNEE and the integrated HASTE extensions allow to abstract nearly all hardware specific details. All adjustable parameters within the HASTE specific GUI determining neuronal or synaptic time constants or specific periods (e.g. refractoriness) can be configured in milliseconds. This means that the user does not have to think about the selected temporal resolution and the consequences arising therefrom for the mapping of the continuous original values to discrete ones.

[1]Doxygen is an open-source tool that extracts specific source code (among others: C++) comments and creates a comfortable HTML documentation

A list of the most important parameters that determine the configuration of HASTE neurons is given in the following info box:

Adjustable parameters provided by HASTE

- Temporal resolution [network cycles/simulated second]
- Number of bits per synapse (ρ_{amp})
- Number of bits used for one single conductance course amplitude
- Shape of conductance course
  - According to suggestions by Maass [15]
  - According to suggestions by Shelley [30]
- Time constant of conductance course for excitatory synapses (τ_{exc}) [ms]
- Time constant of conductance course for inhibitory synapses (τ_{inh}) [ms]
- Transmission delay for all feedback connections (d_{FB}) [network cycles], only 0, 1, 2 and 3 possible
- Refractory period for all neurons, t_{ref} [ms]

Due to its larger flexibility, SoftHASTE provides some additional and more specific parameters, for example separate refractory times for excitatory and inhibitory neurons and an arbitrary transmission delay.

A very important feature of the software created for HASTE is the *I/O visualizer*. During the application of an input sequence the network in- and output can be recorded and visualized in a separate window. For SoftHASTE it is even possible to display the output courses of all CC generators. This does not work for the hardware generated conductance courses, since they cannot be read out. Fig. 2.10 shows an example of this I/O visualizer as it appears in SoftHASTE.

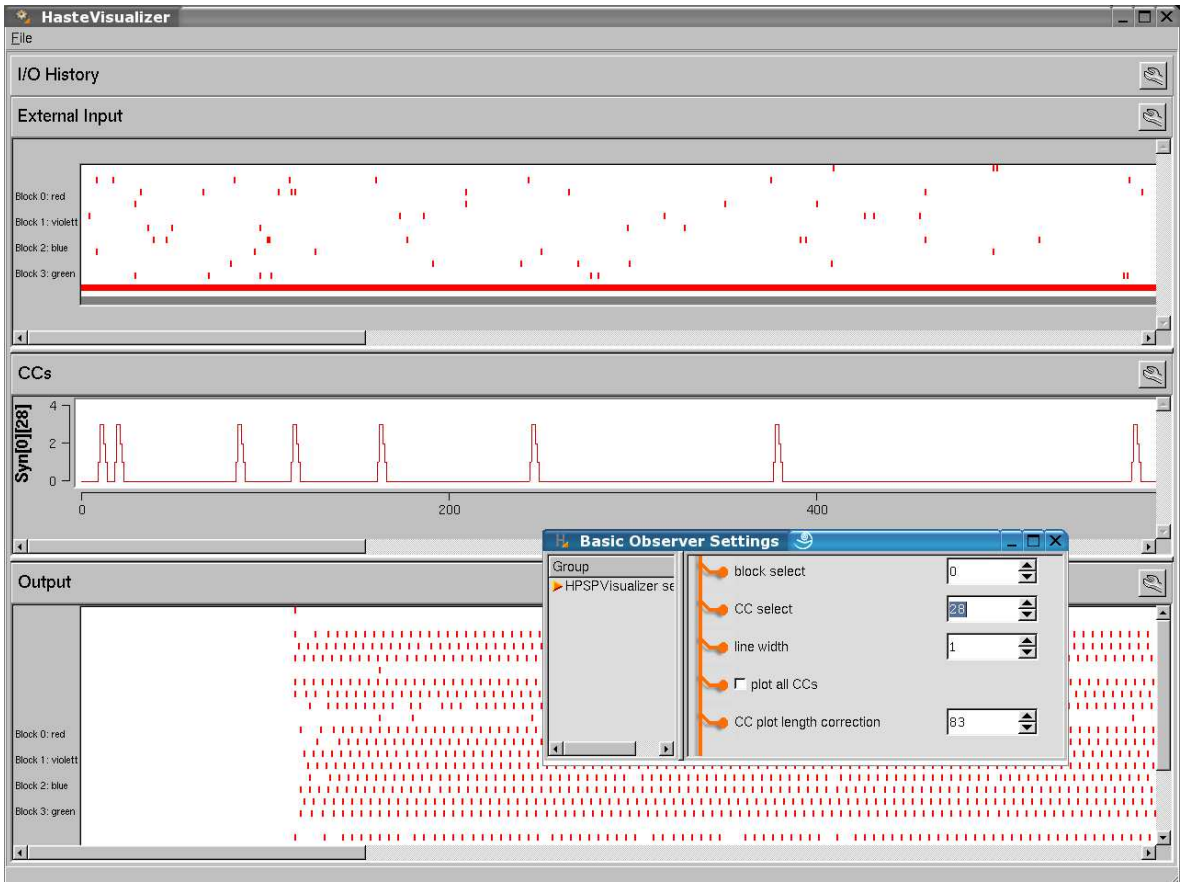


Figure 2.10: Screen-shot of the HASTE visualizer. The large window in the background has three sub-windows. The upper one displays the input injected into the network. Every small red bar denotes an input bit being 1. Each row is another pseudo spike train. Input for different network blocks is colored differently, but since only one network block is used in this case, all trains are red. The second sub-window contains the output of one software CC generator vs. time in ms. The conductance courses can be seen clearly. Obviously they occur in response to the first input spike train. The small window in front of the large one allows to select the displayed CC generator by its block and number. The third sub-window in the large window shows the output of the network. Each row denotes one single neuron's output.

2.2 Methods

2.2.1 Network Generation and Mapping to Hardware

HASTE has been designed with the intention to simulate cortical neurons. Nonetheless it provides a huge configuration space in order to provide maximum flexibility and to find reasonable operating points. A finite region in parameter space is demarcated by typical values found in biology.

Neuron parameters according to Maass

The setup of some experiments presented within this study (see sec. 3.2.3) was strongly oriented towards the liquid computing experiments proposed in [15] by Wolfgang Maass. Although the latter applied an I&F neuron model simulated merely in software, a lot of parameters can directly be conveyed from Maass' setup to the HASTE system. In [15] neuron and synapse parameters are selected to fit measurements in rats' somatosensory cortex best. The refractory period for excitatory (inhibitory) neurons is given by $t_{ref}^{exc} = 3\,\text{ms}$ ($t_{ref}^{inh} = 2\,\text{ms}$). As already mentioned above, HASTE unfortunately implements only one global refractory period, at least for the time being. Hence, a parameter between the two given numbers should be selected. In the referenced work, the postsynaptic current is modeled as an exponential decay, $I_{postsyn} \sim \exp(-t/\tau_s)$. This plain shaping will be used in some experiments to follow (see sec. 3.1.1).

The time constant of this decay is given by $\tau_s^{exc} = 3\,\text{ms}$ ($\tau_s^{inh} = 6\,\text{ms}$) for excitatory (inhibitory) synapses. Maass proposes two different values for the interneuronal transmission delay. For connections from excitatory to excitatory neurons it is set to $t_{trans}^{EE} = 1.5$ ms, for all others it was chosen to be $t_{trans} = 0.8\,\text{ms}$. Again this diversity of time constants cannot be reproduced by HASTE due to the fact that it only realizes one global transmission delay.

Cortical circuits according to Maass

In [15] Maass also gives detailed instruction of how to generate network topologies oriented to biological circuits. Due to the fact that the feedback capabilities of HASTE are limited, mapping networks according to Maass to the available hardware is done as follows: First an abstract three-dimensional network is generated purely in software. The generation is performed by an object of the class *HHasteMaassNetGen*, which holds all necessary and adjustable parameters determining the topology, connectivity and input of the network. The network itself is represented by an instance of the class ThreeDNet, which provides manipulation and access to all information via comfortable public routines. Up to this point no restrictions had to be considered. But mapping this virtual three-dimensional representation of a network to the available hardware neurons is a challenging task. Assume a given amplitude resolution ρ_{amp} for the synapses of HASTE. Then the number of provided neurons $N_{real}(\rho_{\text{amp}})$ is defined by eq. 2.2. The first condition for a proper mapping to the hardware is that the number of abstractly represented neurons N_{virt} is smaller or equal to the number of neurons,

$$N_{virt} \overset{!}{\leq} N_{real} \quad . \tag{2.5}$$

Otherwise distortion would be too strong, since some neurons would have to be skipped. For example the ratio of inhibitory to excitatory neurons would be difficult to keep, or some

parts of the network could even get isolated from the rest. The user interface therefore notifies an error in case of N_{real} being too large. The user has to change the configuration.

If eq. 2.5 is fulfilled, there still can be a discrepancy between the demanded feedback connections and the available possibilities. A straight forward approach is to piecewise map virtual connections to real wires until the hardware is completely occupied. But skipping the remaining parts of the network possibly favors some areas within the three-dimensional representation of the net and discriminates others. Again the danger of isolating parts of the network from the rest arises. Thus the feedback connections that are dropped have to be selected carefully. If a signal is fed into a single input pin of a HAGEN network block, it easily can be connected to more than one output neuron located on this block by configuring the weight array. Usually a single CC generator provides input shaping for more than one synapse, i.e. more than one weight in the columns connecting the generator with the output neurons is unequal to zero. The principle considered to yield best fairness for the skipping of feedback connection is the following: If many neuron outputs compete for a distinct hardware synapse, i.e. for the input of a CC generator, than the probability for a candidate to win the pin is proportional to the number of synapses this generator has to supply.

The following box outlines the mapping algorithm implemented by the method *mapHard()* being part of the class *HMappingHaste2*. This algorithm usually was applied when networks generated according to Maass' instructions had to be mapped to the available hardware.

Mapping an abstract network to the available hardware

- Given the vector $\boldsymbol{n}$ representing all neurons in software
- Given b network blocks with 64 hardware neurons each
- If the size of $\boldsymbol{n}$ is larger than $b \cdot 64$: Notify error, stop!
- Sort $\boldsymbol{n}$ to obtain inhibitory neurons first, then excitatory $\rightarrow \tilde{\boldsymbol{n}}$
- Fill up blocks rotatively with elements of $\tilde{\boldsymbol{n}}$
- For every input of every CC generator (pin): Count competing input signals
- For all external inputs:
  - If an appropriate pin is completely free: take it, done!
  - Else: Randomly select start pin, randomly select direction for search
  - While selected pin already occupied by another external input:
    - Consider pin next to this in selected direction
  - If all pins have been tested and no one was free: Notify error, stop!
  - Else: Found an appropriate pin, take it, done!
- For every input of every CC generator (pin):
  - If external inputs take priority over feedbacks and if there is an external input competing for this pin: Gets it!
  - Else: Select winner randomly with respect to the number of synapses each input represents on this block
  - Update flags and data containers for hardware configuration

The model (and its software representation) locates all somata on the integer points of a three-dimensional grid in space. The Euclidean distance $D(a, b)$ between two neurons a and b determines the probability of a connection $a \rightarrow b$,

$$P_{a \rightarrow b} = C \cdot \exp\left(-D^2(a, b)/\lambda^2\right) \quad . \tag{2.6}$$

Thus it is more likely for somata with a small spatial distance to be connected than for those with large distances. The parameter C depends on the types of neuron a and b, i.e. if they are excitatory (E) or inhibitory (I). Maass suggests values of $C_{EE} = 0.3$, $C_{EI} = 0.2$, $C_{IE} = 0.4$ and $C_{II} = 0.1$. The parameter λ determines the degree of connectivity for the whole network. Maass selected λ to be 2.0. Fig. 2.11 shows a spatial grid with connections generated according to Maass' instructions.

Maass also proposes values for the synaptic efficacy $A(P_1P_2)$ of connections from a neuron of type P_1 to a neuron of type P_2, $P_1, P_2 \in \{E, I\}$. In the I&F model applied in [15] this

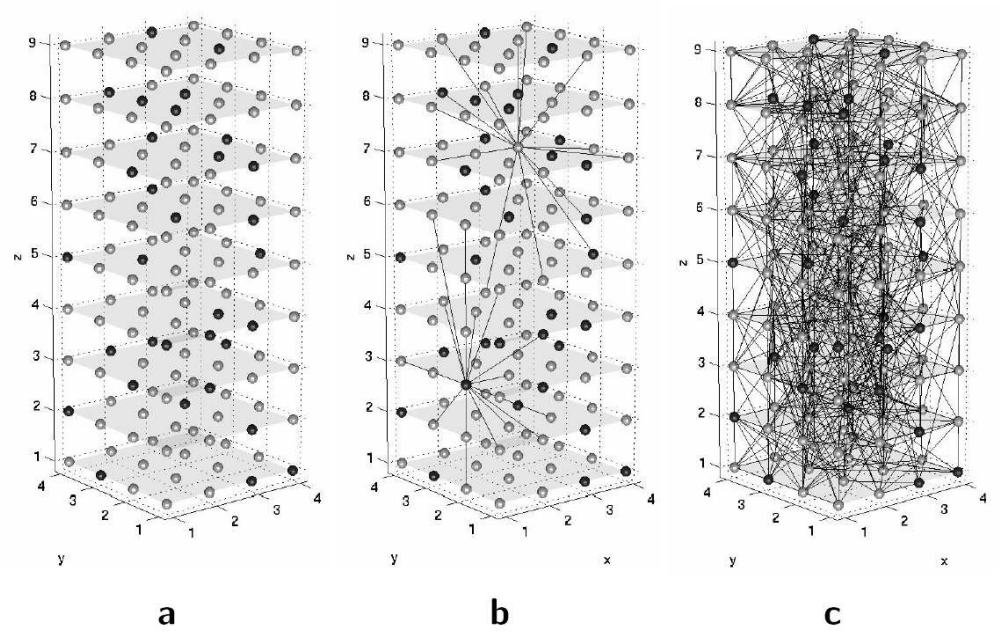


Figure 2.11: Neural network topology according to Maass. The somata are located on the integer points of a three-dimensional grid in space (a). The probability of two neurons to connect to each other depends on their spatial distance and on their types, i.e. if they are excitatory or inhibitory (see eq. 2.6). (b) shows the connection cloud of two selected neurons. (c) illustrates a network generated according to eq. 2.6 with all of its connections. Figures taken from [15].

efficacy denotes the relative amplitude of the exponentially decaying currents the synapses respond with in case of arriving action potentials. Average values of $A_{EE} = 30\,\mathrm{nA}$, $A_{EI} = 60\,\mathrm{nA}$, $A_{IE} = -19\,\mathrm{nA}$ and $A_{II} = -19\,\mathrm{nA}$ and a standard deviation $\sigma = A$ are suggested.

Because of the limited amplitude resolution a CC generator has at its output, the maximum amplitude produced by this device should always equal the largest integer value that is codable with the available bits. Therefore, trying to apply the suggested values of A to the generator amplitudes makes no sense. Another reason supporting this claim is the fact, that a CC generator can supply both excitatory and inhibitory neurons at the same time. Hence, the type of connection cannot be decided at the generator already.

The configuration of the HAGEN weight array defines which CC generator is connected to which output neuron and which synaptic strength this connection exhibits. The weight values have a resolution of 11 bits and therefore easily can be scaled. If the suggestions for A shall be used for a realistic scaling of the simulated conductance courses, one must consider that the spiking of the neuron is not determined by the sum of conductances but by the ratio.

Consider a HASTE neuron, i.e. an output neuron on the HAGEN chip receiving shaped and differently weighted inputs from conductance course generators. Some of these generators modulate input coming from excitatory, others from inhibitory neurons. Let $S_E(t)$ be the sum of all courses arriving from excitatory neurons at a given time t and let $S_I(t)$ be the

corresponding sum for inhibitory inputs. The output of the HAGEN neuron exhibits 1 if $S_E(t) > S_I(t)$, otherwise it is 0.

Hence, the goal is to scale the synaptic weights in a way that the output neuron only fires if the condition given by eq. 1.16 is fulfilled. Therefor the efficacy of connections with an excitatory presynaptic neuron must be decreased by the factor q_{crit}, which is defined in sec. 1.2.5. Regarding this transformation and the suggestions for A made by W. Maass, a synaptic weighting $\tilde{A}$ can be proposed for the usage with HASTE that results in a biologically inspired relative efficacy of different connection types:
$\tilde{A}_{EE} = 0.499\,\text{hwu}$, $\tilde{A}_{EI} = 1.000\,\text{hwu}$, $\tilde{A}_{IE} = -0.887\,\text{hwu}$ and $\tilde{A}_{II} = -0.887\,\text{hwu}$. In practice the effective total weights of multi-bit synapses (see fig. 2.9) were drawn with a certain standard deviation around their corresponding mean values. Due to the limited range of applicable weights on the HAGEN chip the suggested standard deviation of $1.0A$ of the mean had to be reduced to a constant value of 0.1 hwu.

Maass suggests to randomly select 20 %of the neurons to be inhibitory, the rest excitatory. This corresponds to distributions found in real cortical networks.

Since for nearly every network generated according to this instruction with realistic parameter values the number of available feedback connections is insufficient, the mapping algorithm described above was an important tool to avoid distortion of the topology character when skipping some synapses.

2.2.2 Input Patterns

The usual way to run experiments on the HAGEN chip is to apply sequences of bit vectors to its inputs and possibly enable feedback of the neurons' resulting outputs. These input sequences have to be generated by the controlling software, i.e. by some modules of HANNEE. Before HASTE was realized there already was a bunch of input pattern generators included in the default HANNEE setup.

Poisson spike trains

One of the most important for the experiments to follow is a generator for high-dimensional bit streams with randomly Poisson distributed bits being 1. The term *Poisson distribution* allows for the probability $p(t_1 - t_0)$ of two subsequent 1-bits occurring at cycles t_0 and t_1. These active bits can be interpreted by HASTE as spikes (see sec. 2.1), Poisson distributed bit streams represent *Poisson spike trains*. This kind of spike distribution is commonly considered to be a good estimation for the activity of cortical neurons, see [30]. Since such a randomly generated bit stream does not contain information it is usually utilized for the simulation of unspecific input or background. The Poisson spike train generator can be set to produce an arbitrary frequency of spikes.

Random bit trains

Another type of bit stream generation that came into operation is the so-called *Random bit train*: Every bit is set to 1 with a certain probability p_{on}, randomly and independently from all other bits. This type of bit stream with $p_{\text{on}} = 0.5$ is best suited for parity tasks.

Modulating input patterns

Due to the necessity of speeding up or down the information flow into a network, for some HASTE experiments it was necessary to apply every single bit of a bit stream multiple times, i.e. to blow up the stream in time domain. Therefore a new interface class, namely *H?*, was written. This class implements the growing of arbitrary bit streams by a selectable integer factor b. A new bit stream is generated by repeating every bit of the original stream b times. This will be referred to as *temporal blowup* of a bit stream. Another modification is simply stretching a stream, i.e. filling an arbitrary number of zeros between every pair of subsequent bits. In what follows this will be called *temporal blowup with zeros*.

2.2.3 Utilizing MATLAB

MATLAB is a software product for numeric computation that is optimized for linear algebra, i.e. particularly for matrix operations. It also is a powerful tool for the convenient visualization of data. It has been applied on most of the raw experimental data retrieved for this study in order to evaluate statistical errors and to plot the results in a clearly arranged manner. In appendix A.2 the path where to find the sources is given.

Chapter 3

Experiments

The hardware implementation of HASTE was preceded by the creation of a pure software version capable of exploring design variables, called SoftHASTE (see sec. 2.1.3). SoftHASTE easily can switch parameters like synaptic amplitude resolution or the way of superposing conductance courses (CCs) at one synapse. It can apply arbitrary and diverse CCs for every single synapse. Thus in the beginning SoftHASTE was used to explore and cut down a huge parameter space, preparing and supporting the realization of HASTE on the FPGA in the long term.

The complexity of information processing performed within the experiments increases during this study. In the *first section* of this chapter methods were developed for HASTE to allow resolution variations without affecting output firing rates.

Although rate coding is very often considered to be the main mechanism of information transmission in biological neural networks, it can not be the only way to convey information. Rapid processing of visual stimuli in the brain for example was proven not to be compatible with rate coding [33]. The concept of temporal coding can explain many features that are not covered by rate coding [22]. Hence in the *second section* an important condition for temporal coding is tested on HASTE: The diversity of interspike intervals.

The *third section* of this chapter deals with the application of *liquid computing*, which was introduced in sec. 1.3.

3.1 Basic Studies

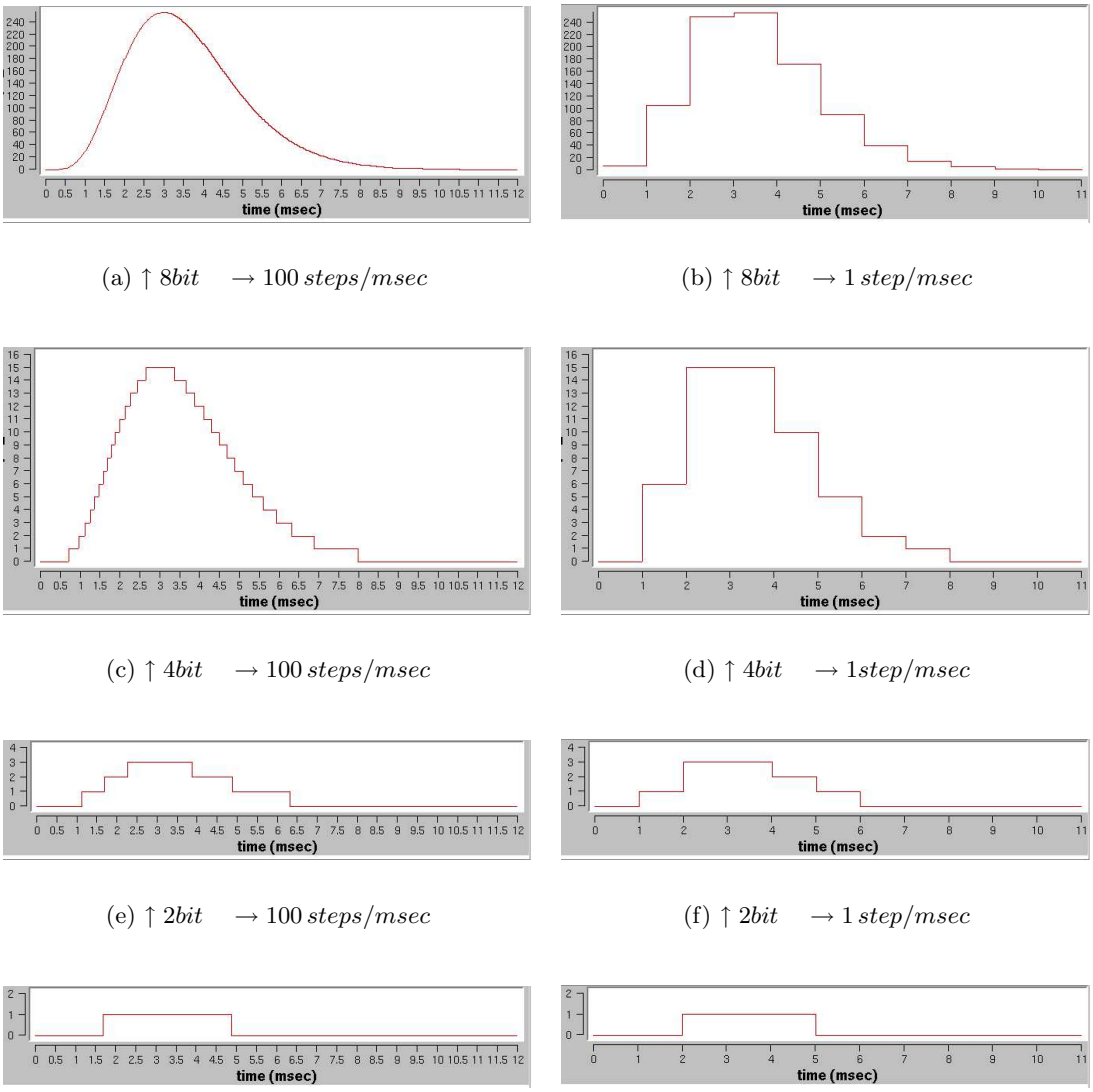
One has to cope with the fact that the HAGEN system is able to work only in discrete time domain and that the chip provides just a finite number of neurons, synapses and feedback possibilities. Due to these limited hardware resources the number of available neurons for the HASTE extension had to be reduced by the factor n when pooling ρ_{amp} binary inputs to one multi-bit unit (see sec. 2.1). To gain network topologies of reasonable complexity the synapse resolution therefore had to be chosen low. This limited amplitude resolution of simulated conductance courses and the finite temporal resolution ρ_{temp} are obvious restrictions of the HASTE system. As an example, fig. 3.1 illustrates the effect low resolutions can have on a CC shape. The original conductance course these figures are based on is described by an equation given further below in this section (eq. 3.1). The exact discretization method applied to the continuous shape is an essential point of this work. It will be the subject of a subsection to follow.

The main goal of the HASTE approach was and still is to find out which characteristics of synaptic behavior are essential for certain capabilities of the network like memory capacity or self organization. In other words: How much precision for modeling a conductance course is necessary to obtain a certain network feature?

Two design variables of HASTE dominate the accuracy of the simulation: ρ_{amp} and ρ_{temp}. An important feature of HASTE is the *adjustable* synapse resolution. Nonetheless results of experiments with different shape resolutions must be comparable quantitatively. Consequently a criterion to match the CCs modeled with different resolutions is required. Such a criterion is described in sec. 3.1.1.

New questions arise from the possibility of input shaping by increasing synaptic amplitude resolution. The function describing the temporal course of membrane conductance initiated by an incoming spike is far from being indisputable and will be studied within this work. The time delay of interneuron connections and especially the diversity of delays may also have an impact on network capabilities.

It was a challenge to sensibly balance the flexibility and range of all required parameters. The following experiments were intended to coarsely give an idea of how the system works with different setups, mainly regarding temporal and CC amplitude resolution. They were performed utilizing SoftHASTE.



(a) ↑ 8*bit* → 100 *steps*/*msec*

(b) ↑ 8*bit* → 1 *step*/*msec*

(c) ↑ 4*bit* → 100 *steps*/*msec*

(d) ↑ 4*bit* → 1*step*/*msec*

(e) ↑ 2*bit* → 100 *steps*/*msec*

(f) ↑ 2*bit* → 1 *step*/*msec*

(g) ↑ 1*bit* → 100 *steps*/*msec*

(h) ↑ 1*bit* → 1 *step*/*msec*

Figure 3.1: CC shapes for different amplitude (ρ_{amp}) and temporal (ρ_{temp}) resolutions as they are produced by HASTE. They are based on a continuous course defined by eq. 3.1. τ was set to be 0.6 ms, ρ_{temp} is 100 steps/ms for all plots in the left and 1 step/ms for the plots in the right column. The amplitude is scaled to the maximum integer value that can be achieved with the available bits. In all cases the translation from continuous to discrete values was done with arithmetic rounding. The target shape is well reproduced in (a).

3.1.1 Single Neuron Bombardment

Michael Shelley et al. used biologically inspired spike trains with Poisson distributed interspike times for impinging their simulated network in order to put it into High Conductance State, see sec. 1.2.5 and [30]. The work presented here advances McCulloch-Pitts neurons with binary output towards more biologically realistic behavior. Hence the applied criterion for comparability of different synapse resolutions included a biological characteristic: A *HASTE neuron bombarded with Poisson spike trains* was claimed to fire with a distinct average rate independently from the number of bits used for CC modeling (see fig. 3.2 for a schematic). The goal of the experiments to follow was to find a discretization method which fulfills this condition. Additionally a minimum temporal resolution had to be found in a way that increasing this resolution would not change the response behavior of bombarded neurons significantly any more.

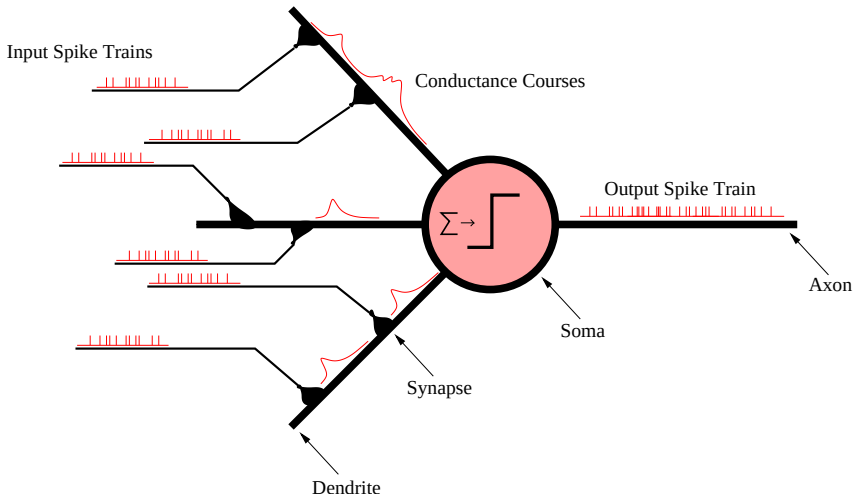


Figure 3.2: Schematic of a *single neuron bombardment* experiment: One neuron is impinged with several Poisson spike trains. Each synapse answers a spike with a change of the soma's membrane conductance (simulated in software, later on FPGA). All conductance courses are summed up along the dendrites and in the soma. HASTE utilizes the HAGEN chip for this superposing. The output neurons on HAGEN discriminate if the excitatory-to-inhibitory conductance ratio is large enough to make the simulated effective reversal potential (see sec. 1.2.5) cross the spiking threshold.

Basic setup

For the first experiments performed with HASTE, all of them being single neuron bombardments, the full and HAGEN chip with calibrated synaptic weights (see sec. 2.1.1) was used. This means that all available neurons on the chip received the same input, but worked independently from each other, i.e. without any interconnection. As the hardware neurons were not expected to respond exactly identically to the same input, the simultaneous operation of as many of them as possible provided statistical information about mean and deviation of the quantities observed.

Every output neuron was impinged with 30 spike trains, which is the largest input number one can realize with the planned FPGA version of HASTE on one HAGEN block, using 4-bit CCs. This can be explained as follows: One network block on the HAGEN chip provides 128 input pins. 120 of them are occupied by the 4-bit conductance course generators. One more is skipped for chip specific reasons (the pin closest to the edge of the chip, see sec. 2.1.1), two more are occupied by calibration inputs and another one is used for a bias input which adjusts the threshold. For 2-bit (1-bit) CCs the number of used input pins per block decreases to 60 (30). Statistical data for one distinct hardware neuron was collected by averaging output firing rates over intervals of 3 sec simulated real time.

Every input spike train had an average firing rate of 33 Hz, the spike times were Poisson distributed. In biological context 33 Hz is a high firing rate, regular cortical activity in awake state happens in the order of 5 Hz. But since only 30 inputs were possible, which is at least a factor 20 too low to be realistic, a total input producing sufficient activity had to be created by choosing such a high input rate per synapse. With the values selected a mean total input spike rate of $\sim 1\,\mathrm{kHz}$ is achieved. All inputs were injected with the same synaptic weight. This was swept from 0.0 hwu to 1.0 hwu, i.e. all inputs were considered to be excitatory. The neurons' refractory time was set to 3 ms.

Threshold

Setting the threshold to a value larger than zero is essential for a neuron impinged with excitatory spikes only. If there was no threshold the neuron would fire at its maximum rate for the duration of synaptic response to each single input spike. Compared to nature this would be much too sensitive. A real neuron's output spike is triggered only after about 10 to 100 input spikes within a short time window. In case of a negative spiking threshold configuration the HASTE neuron would fire even endlessly.

As already mentioned in the theoretical part of this work (see sec. 1) a cortical neuron has 10^2 up to 10^4 inputs. Hence in the scope of a neuron population in High Conductance State the threshold of a McCulloch-Pitts neuron can be interpreted as the total conductance induced by many other inputs that can not be simulated in detail (see section "A new way of simulation" on page 13 and especially eq. 1.15). In the scope of a single cell the threshold implemented in HASTE carefully can be compared with the voltage to be crossed by the membrane potential of a real neuron for initiating an action potential. This analogy is not absolutely correct since HASTE simulates membrane currents induced by conductance changes, it does *not* simulate membrane potentials and PSPs. To adjust the threshold for all experiments to follow a constantly active input was applied to every neuron. its weight was set to -1.0 hwu, i.e. to make the neuron fire the sum of all inputs multiplied by their weights had to exceed 1.0 hwu.

Shaping

The synapses generated conductance courses according to suggestions found in [30], i.e.

$$G_i^{\mathrm{Shelley}}(t) \sim \frac{(t-t_i)^5}{\tau_\sigma^6}\, exp\left(-(t-t_i)/\tau_\sigma\right)\Theta(t-t_i) \qquad , \qquad \sigma = E, I. \tag{3.1}$$

$\Theta(t)$ is the unit step function, t_i denotes the arrival time of spike i and E (I) means that the simulated synapse is excitatory (inhibitory). In case of many spikes running into the same synapse within a short time window an important question was how to manage temporally

overlaying $G_i(t)$ *at this synapse*. For the experiments presented in this section the component of a neuron's total membrane conductance provided by its synapse j was obtained by superposing all this synapse's spike answers,

$$g_j^{superpos}(t) = \sum_i G_i(t) \qquad . \tag{3.2}$$

Here i indexes all spikes impinging the synapse j. The way of managing overlapping CCs shall be universal for all shapes. Therefore the superscript *Shelley* has been skipped. If this superposed conductance component exceeded the maximum value representable with a multi-bit synapse then the synapse's maximum value was applied instead.

Another possibility to cope with temporally concurring CCs at one synapse would be a simple reset each time a new spike arrives, i.e. only the most current shape is transmitted. Let $\hat{I}(t)$ be the index of the last spike that occurred at synapse j. Then the conductance at synapse j can also be described by

$$g_j^{reset}(t) = G_{\hat{I}(t)}^{\text{Shelley}}(t) \qquad , \tag{3.3}$$

The neuron's total membrane conductance is the sum over all synaptic components,

$$g^{tot}(t) = \sum_j g_j(t) \qquad . \tag{3.4}$$

This superposing of many components was performed by the HAGEN chip. A HAGEN neuron has an upper limit for its total input current, too, but the input rates and synaptic weights selected for the experiments presented here were much too low to reach this ceiling (see sec. 2.1.1).

The CC shape described by equation 3.1 is well reproduced in fig. 3.1(a). The time constant τ was set to 0.6 ms, which is in biologically realistic ranges [30]. The problem now was how to map this continuous shape to the small number of discrete values provided by a HASTE synapse. Different methods have been tested in single neuron bombardment experiments.

From continuous to discrete values

All mapping methods that have been tested have a basic procedure in common, see fig. 3.3 for illustration. This way of discretization was considered to provide maximum accuracy in keeping the original shape, but leaving parameters to optimize mapping according to different demands. First, for every interval between two discrete time steps k and $k+1$ the original CC shape (1) is averaged over n samples (2), where n is a parameter adjustable via the HASTE GUI (see sec. 2.1.3). This average value is assigned to time step k, thus the CC then is discrete in time(3). The amplitude of this steplike CC, still being a non-integer value, then is scaled to the maximum integer v_{max} that can be represented with the available bits (4,5). Afterwards a constant offset $0 < \delta_{\text{round}} < 1$ is added to the scaled shape (6,7). Finally for every time step the discrete CC value is determined by rounding down the scaled and biased CC to the provided integer values (8,9).

When talking about discretization a common approach is arithmetic rounding (ARN). For digital machines rounding to integers normally is implemented by adding 0.5 to the value to be rounded and then truncate all digits smaller than one. The basic method described

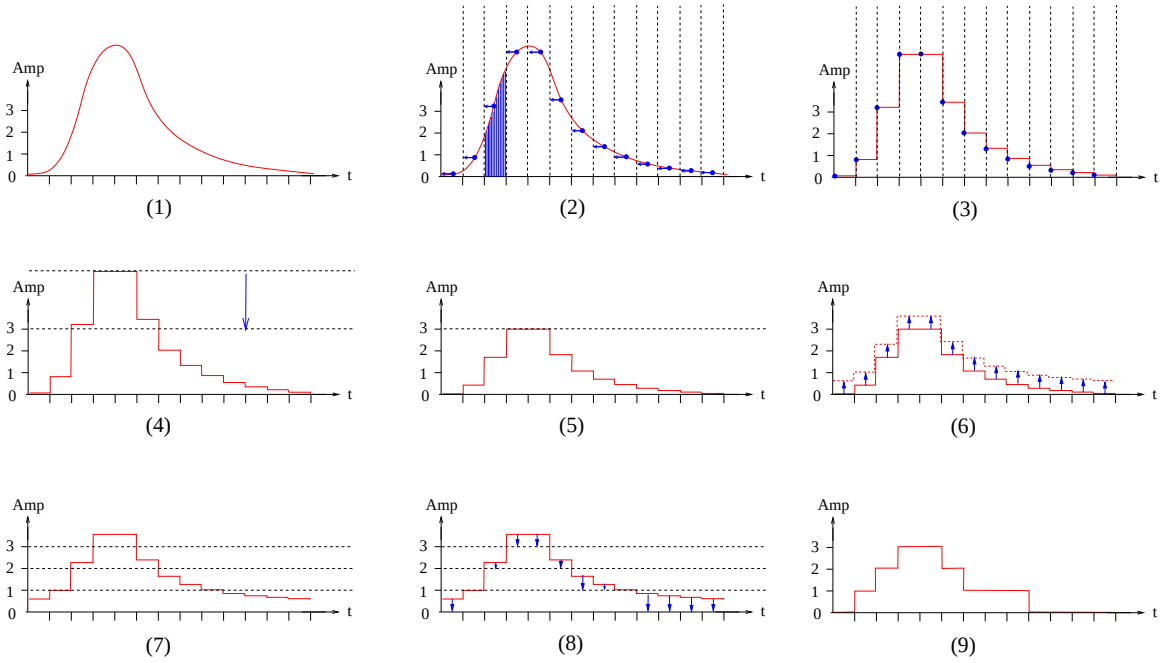


Figure 3.3: Process of shape mapping from continuous to discrete values, see text for details.

above performs ARN by setting δ_{round} to 0.5. The CCs shown in fig. 3.1 have been created according to the suggested procedure and with a constant rounding offset of $\delta_{\text{round}} = 0.5$.

As the effect of the CCs on the output firing rate was to be kept constant a second idea was to dynamically optimize the rounding offset δ_{round} in a way that the integral over time of a conductance course (with normalized amplitude) is always as close to the integral of the original shape as possible. The term *dynamically* means that for every bit resolution another rounding offset can be applied. This method will be called CCI for *Constant Course Integral*. A critical scenario for both ARN and CCI is a conductance course resolution of only one bit: Regarding a realistic CC shape with a strongly localized peak and a long drawn-out tail arithmetic rounding on the one hand might result in a 1-bit shape much too short. Arithmetic rounding truncates already when the amplitude falls below half of the peak value, but the peak itself is reproduced in a way. Keeping the integral constant on the other hand ignores the original peak and thus might deliver a shape too long to support coincidence detection properly, for example. Also conceivable are constant rounding offsets δ_{round} smaller or larger than 0.5, which possibly can avoid the extreme cases described above.

Measurements with different temporal and amplitude resolutions

The first experiment performed with HASTE was a single neuron bombardment with settings as described above (subsection "Basic settings"). Conductance courses were discretized to the available integers via arithmetic rounding. For three different bit resolutions – namely one, two and four – the mean output firing rate was measured with temporal resolutions of 1000, 2000, 4000 and 8000 steps/sec. Resolutions lower than 1000 steps/sec do not make too

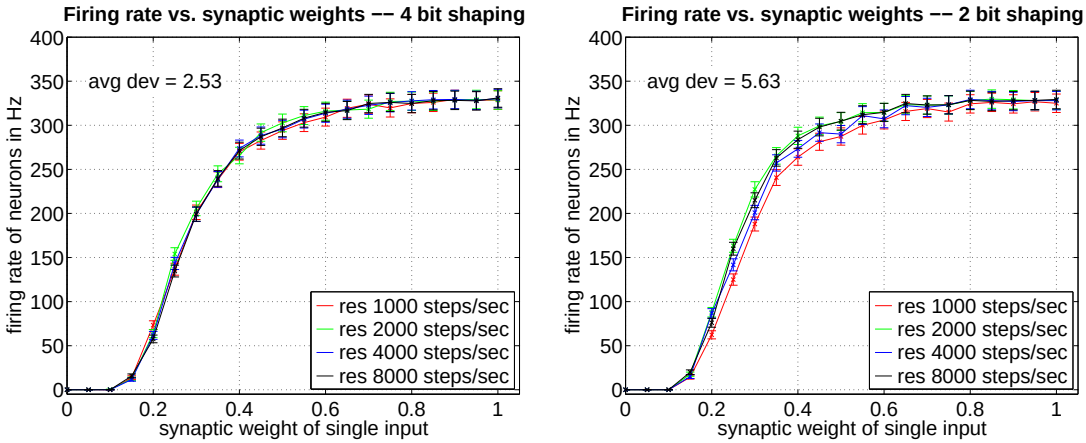
much sense as basic neuronal time constants are in the order of 1 ms. Typical values for absolute refractory times for example are $2 - 5$ ms, action potentials take about 1 ms, time constants of decaying synaptic signals are in the order of 3 ms. The goal of this first experiment was to detect dependencies of a neuron's output firing rate on both the number of bits per synapse and temporal resolution.

To start with, fig. 3.4 shows the results recorded for just *one single* neuron. The error-bars represent statistical error caused by the finite time of measurement only. Obviously the neuron's output firing rate saturates for high input weights. The upper bound of 333 Hz is given by the inverse of the refractory time, which was 3 ms for this setup. Obviously the weight of -1.0 hwu for the bias input adjusting the threshold was reasonable, because the whole available output frequency spectrum is covered. The curves for different amplitude resolutions are similar, but for the 1-bit curves a steplike course appears. A simple model was developed to explain this behavior, it will be presented further below.

In order to assess the matching of different curves, the following measure was applied (as an example consider fig. 3.4(c)): For a fixed point on the x-axis, i.e. a fixed input weight $\tilde{w}$, four different rate values belonging to four different temporal resolutions are assigned to $\tilde{w}$. Those four values have an mean and a standard deviation. The sum of all standard deviations over all sampled points on the x-axis divided by the number of these points was considered to be an appropriate measure for the degree of curve matching. It is shown in every figure containing more than one curve, labeled as *avg dev*.

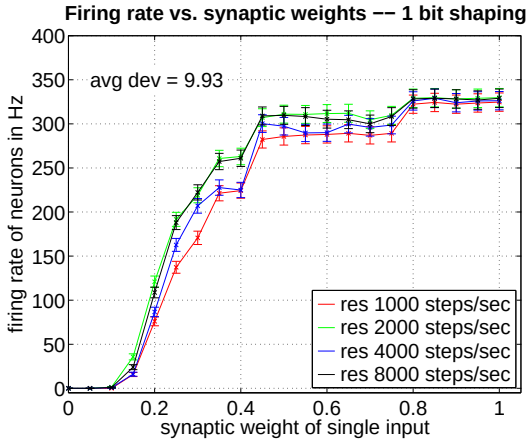
Fig. 3.5 shows the same experiment like fig. 3.4(c), but averaged over all neurons available on the HAGEN chip. This was to include errors possibly arising from hardware specific differences between the neurons and imperfect calibration. Here the error-bars include both the statistical error of each single neuron and the standard deviation of all mean firing rates over all neurons for one distinct input weight. Comparing the one-neuron measurement with the multi-neuron average it can be seen that in regions of high slope the error resulting from hardware heterogeneity dominates the statistical error, thus increasing simulation time was not necessary.

The data retrieved so far shows that for $\rho_{\text{amp}} = 1$ bit and arithmetic rounding, different temporal resolutions lead to significantly different firing rate curves in a way that does not seem to converge for higher values of ρ_{temp}. Hence amplitude resolutions of at least two bits have to be recommended for arithmetic rounding, if a neuron's independence on temporal resolution is important.



(a) 4-bit shapes

(b) 2-bit shapes



(c) 1-bit shapes

Figure 3.4: Output firing rate of a single neuron bombarded with 30 Poisson spike trains versus the synaptic weight of a single input (all 30 inputs have the same weight). Each input spike train had an average firing rate of 33 Hz. All continuous CC shapes were mapped to the available integers via arithmetic rounding. For each amplitude resolution (a, b, c) four different temporal resolutions (colored) are plotted.

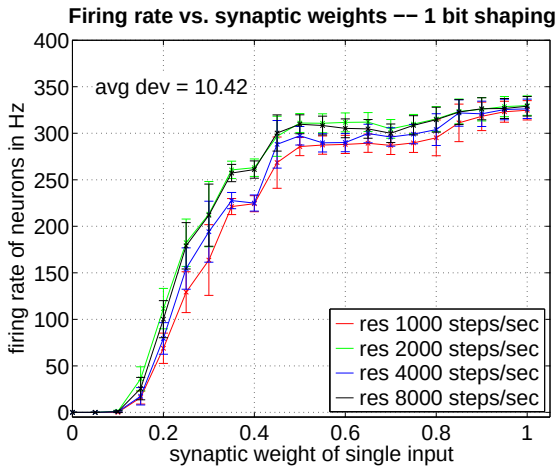


Figure 3.5: Same as fig. 3.4(c), but averaged over a maximum number of neurons available on one HAGEN chip. Each data point shows the mean over 240 independently driven hardware neurons. The error-bars include both statistical error of each measurement and the standard deviation of the mean values over all neurons.

Comparing different discretization methods

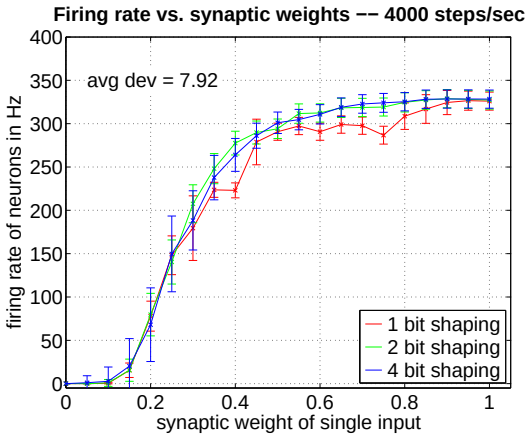
The following experiments – all of them single neuron bombardments – were performed in order to find out which mapping method matches output firing rates of bombarded neurons with different ρ_{amp} values best. The temporal discretization was demanded to have no effect on the rates in a range from 1000 to 4000 steps/sec. Therefore different methods were applied for one, two and four bit amplitude resolutions while the temporal resolution was set to 1000 and 4000 steps/sec.

Fig. 3.6 shows four experiment series with different methods of discretizing continuous conductance shapes. The sub-figures (a) and (c) were obtained by adding a constant rounding offset of $\delta_{\text{round}} = 0.5$ to the shape values, not depending on the number of used bits, before truncating them to the next integer. This has been called the ARN method. In the experiment series shown in the sub-figures (b) and (c) for every bit resolution the rounding offset was varied from 0.0 to 0.99 in steps of 0.01, until the integral of the new shape matched best with the integral of the continuous course. This is the CCI method mentioned above.

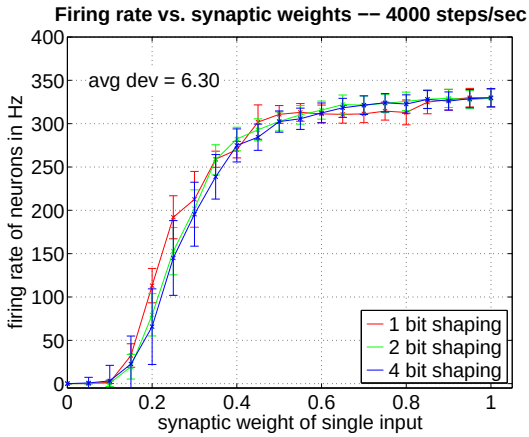
In appendix A.1 another figure is presented (fig. A.2), which shows four experiment series with constant rounding offsets δ_{round} of 0.3, 0.4, 0.5 and 0.6. There the shift of the curves relative to each other depending on the value of d can be observed.

Qualitatively, for both temporal resolutions the CCI method seems to match the three curves best. This result is supported by the smallest *avg dev* value for CCI. The methods using constant rounding offsets achieve best overall matching for $\delta_{\text{round}} = 0.6$, at least according to the measure *avg dev*. But as arithmetic rounding ($\delta_{\text{round}} = 0.5$) delivers much better matching for low output firing rates ($f < 100\,\text{Hz}$, this is the biologically realistic region) and because of its mathematical persuasiveness it might be preferred.

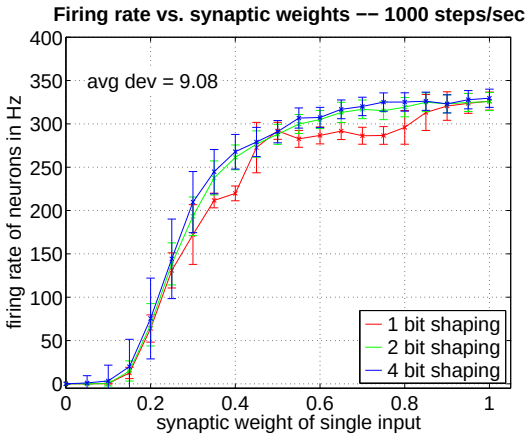
Especially because of the steplike curves obtained with low amplitude resolution a perfect matching of the firing rate curves is not possible. But the deviations caused by both ARN and CCI seem to be dominated by hardware specific errors, which makes them both candidates for application. The overall matching delivered by CCI is best in the experiments presented, but for low output frequencies ARN may be the preferred choice.



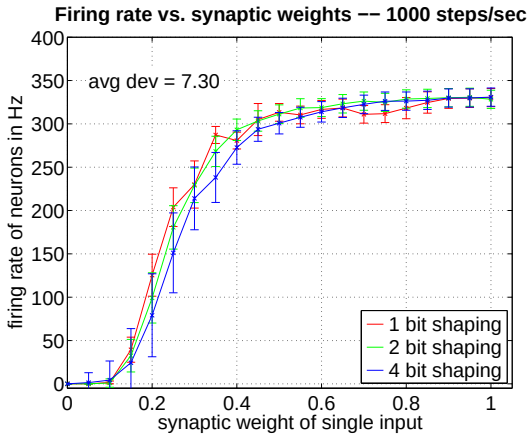
(a) ARN, $\rho_{\text{temp}} = 4000\,\text{steps/sec}$



(b) CCI, $\rho_{\text{temp}} = 4000\,\text{steps/sec}$



(c) ARN, $\rho_{\text{temp}} = 1000\,\text{steps/sec}$



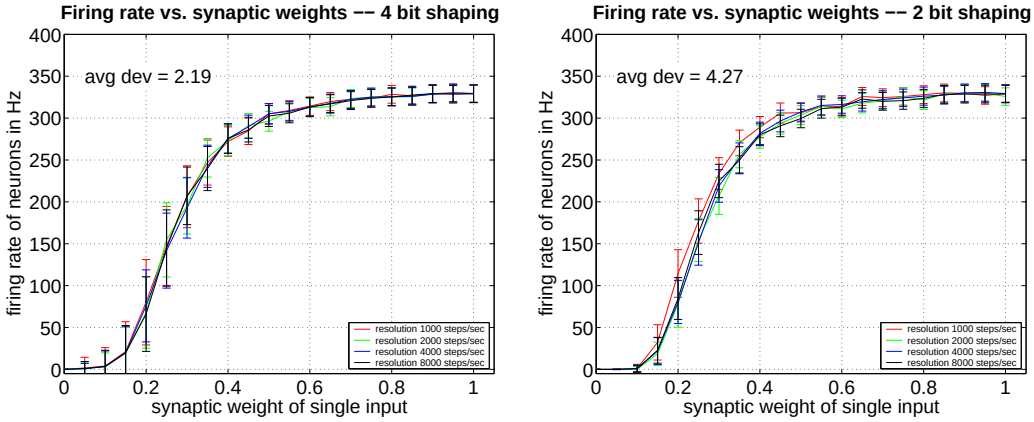
(d) CCI, $\rho_{\text{temp}} = 1000\,\text{steps/sec}$

Figure 3.6: Output firing rate of a single neuron bombarded with 30 Poisson spike trains versus the synaptic weight of a single input (all 30 inputs have the same weight). Each input spike train has an average firing rate of 33 Hz, the temporal resolution is 4000 steps/sec for (a) and (b) and 1000 steps/sec for (c) and (d). (a) and (c) were obtained with a mapping offset $d = 0.5$, i.e. with arithmetic rounding (ARN). For (b) and (c) the mapping offset was optimized to keep CC integral constant (CCI).

Varying ρ_{temp} with CCI

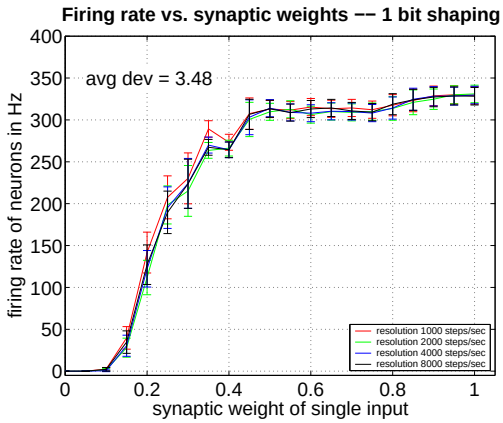
With these insights the dependence of HASTE with CCI mapping on temporal resolution might be interesting. Thus the experiments proposed first in this section were repeated, i.e. single neuron bombardments for different amplitude and temporal resolutions, but now applying CCI instead of ARN. The results can be seen in fig. 3.7.

This experiment reveals a clear advantage of the CCI method: the dependence of a network using conductance courses generated with CCI on temporal resolution is much smaller than for arithmetic rounding. Even for a 1-bit setup 1000 discrete time-steps per second seem to be enough. The experiments performed indicate no advantage of a higher temporal resolution. Hence for experiments where temporal resolution might be a critical parameter for optimization and thus needs comparability the method of keeping shape integrals constant should be applied.



(a) 4-bit shapes

(b) 2-bit shapes



(c) 1-bit shapes

Figure 3.7: Same as fig. 3.5, but the method for mapping from continuous to discrete conductance course is CCI instead of ARN. Each data point in (a) shows the mean over 60 independently driven hardware neurons. (b) corresponds to 120, (c) to 240 neurons. The error-bars include both statistical error of each measurement and the standard deviation of the mean values over all neurons. The matching of the different curves per figure is much better than with the ARN method.

A simple 1-bit model

The distinct step character of the red curves in figures 3.5 and 3.7 can easily be understood by studying the following model:
The bombarded neuron will only fire if the sum of its inputs multiplied by their synaptic weights is above threshold V_{thresh}. In the experiment described here, all input weights w_i were the same, $w_i = w \; \forall i$. For the red curves only one bit was used for conductance course shaping. Therefore a critical number n_{crit} of inputs exists that have to be active in order to exceed the threshold. Roughly speaking, the steps mark the input weights where this critical number n_{crit} changes by the value one. Let p_{sub} be the probability that the number of inputs being active at a given moment is *not* enough to obtain

$$\sum_{i=1}^{N} w_i \cdot s_i \geq V_{thresh} \tag{3.5}$$

where N is the total number of inputs and s_i is the value of input i, $s_i \in \{0, 1\}$.

The maximum firing rate of a neuron is given by the inverse of its refractory time t_{ref},

$$f_{max} = \frac{1}{t_{ref}} \quad . \tag{3.6}$$

The basic assumption of the 1-bit model is that the mean output firing rate $\hat{f}_{out}$ of the neuron can be expressed by p_{sub} via

$$\hat{f}_{out} = f_{max} \cdot (1 - p_{sub}) \qquad . \tag{3.7}$$

For single bit inputs, p_{sub} is easily calculated. Let f_{in} be the mean firing rate of each input and T_{CC} be the duration of a single conductance course being above zero. Furthermore assume $\frac{1}{f_{in}} \gg T_{CC}$. Then for a single input into a neuron the probability to fire is

$$p_{single} = f_{in} \cdot T_{CC} \qquad . \tag{3.8}$$

The number of inputs firing at a given moment is Poisson-distributed, as the regarded system works in discrete time steps.

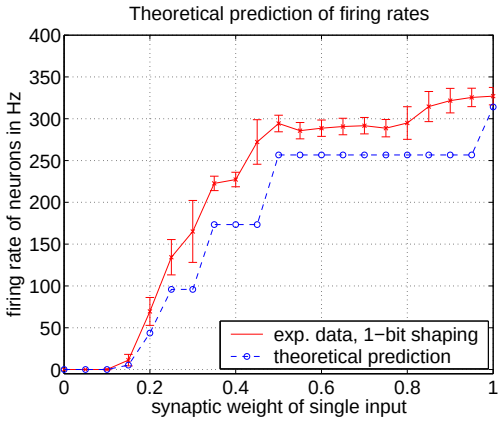
$$p_{sub} = \sum_{i=0}^{n_{crit}-1} \binom{N}{i} \cdot p_{single}^{i} \cdot (1 - p_{single})^{N-i} \qquad . \tag{3.9}$$

As an example, the model was evaluated applying the parameters used in the experiments shown in fig. 3.5 (b), (c) and (d). The resulting predictions are drawn in fig. 3.8 (blue dashed curves). Keep in mind that these predictions only describe the behavior of single bit CCs and that this model does not consider any hardware specific details. Moreover eq. 3.7 is just an assumption for large statistics. The characteristic steps are qualitatively reproduced, though.

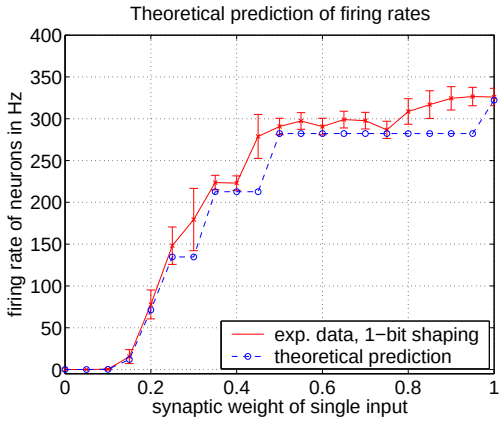
To obtain a better matching with experimental data, a spontaneous input firing rate $p_{spont} = 4\,\text{Hz}$ was added to f_{in} a posteriori. This can be justified by current fluctuations on the HAGEN chip (see sec. 2.1.1) causing the total input driven membrane current to sometimes overstep the threshold spontaneously when already very close to it. Additionally the synaptic weight of the bias neuron has been set 13 % lower than in the experimental setup. This variation optimizes matching, too, and can be explained by chip specific details as well:

As the synapses connecting the bias input with the bombarded neuron have been positioned close to the edge of the HAGEN chip, they have not the same surrounding as synapses positioned more closely to the center. Therefore, parasitic conductances and capacities may cause differences in the effective value of these connections. This already has been observed in other experiments [8]. Both optimizations were applied a posteriori by trial-and-error and are not to be understood as a sophisticated extension of the model. But after these small cosmetics, the position of the steps is accurately reproduced, see fig. 3.9.

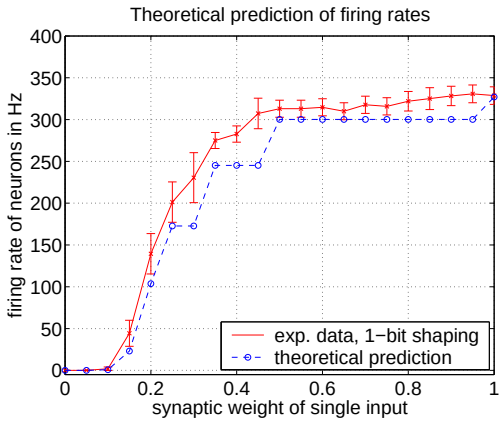
For higher bit resolutions, p_{sub} would be much more difficult to calculate, but as the pursued target is not to mathematically model the behavior of HASTE in detail, this 1-bit example should be enough.



(a) d = 0.4

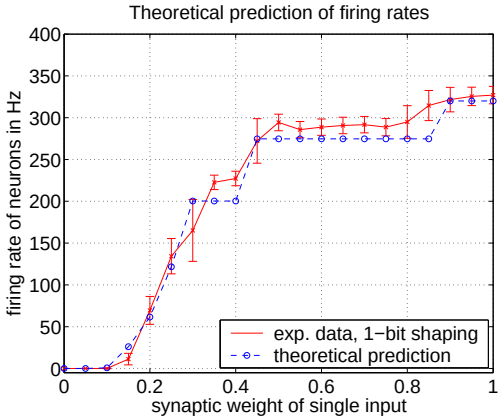


(b) d = 0.5

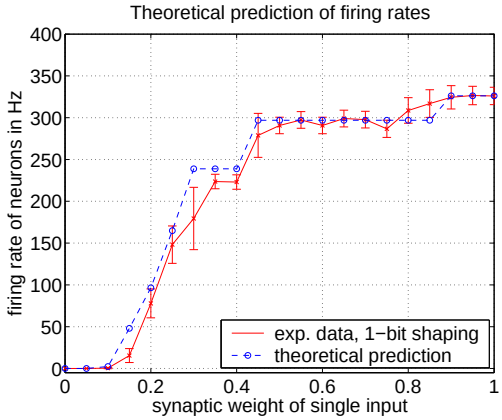


(c) d = 0.6

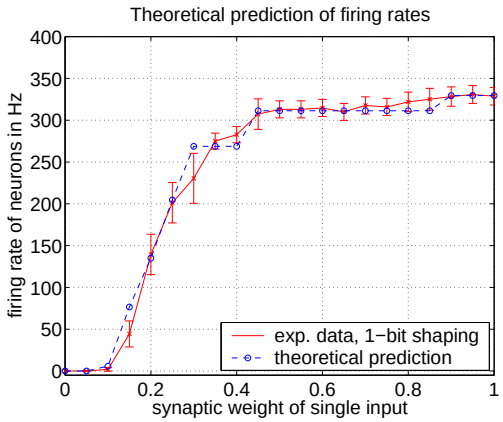
Figure 3.8: Experimental data (red) and theoretical values (blue) as predicted by the model proposed in the text. The experimental data was already presented in fig. A.2, the parameters are mentioned in the context. The steplike course of the curve is already reproduced well.



(a) d = 0.4



(b) d = 0.5



(c) d = 0.6

Figure 3.9: Like fig. 3.8, but with slight modifications of the prediction model inspired by hardware specific details (see text). Now the experimental data (red) is even quantitatively reproduced within the error-bounds for nearly all measured points.

3.1.2 Conductance Course Variation

More experiments have been carried out with conductance courses described by a different equation. The CCs were modeled by

$$G_i^{\text{Maass}}(t) \sim e^{-(t-t_i)/\tau_\sigma}\,\Theta(t-t_i) \qquad , \qquad \sigma = E, I, \tag{3.10}$$

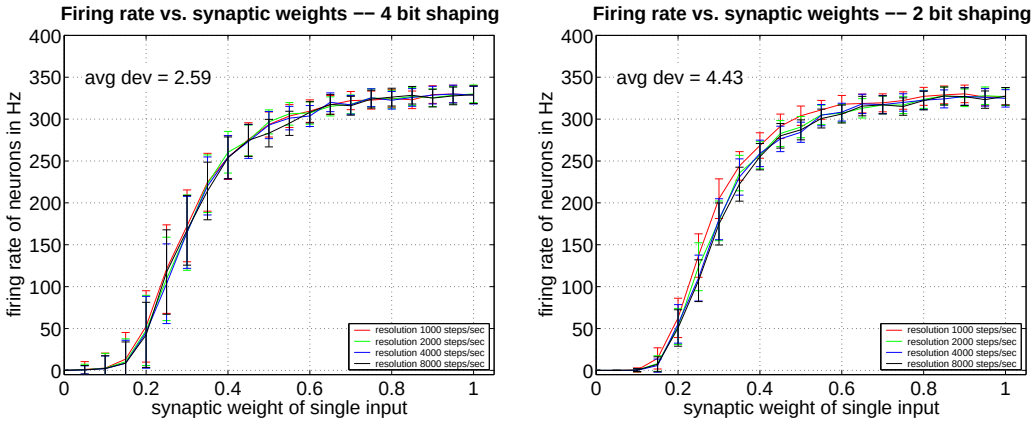
which is a simplification of eq. 3.1 and was suggested, among others, by Wolfgang Maass [16]. E, I, σ, $\Theta(t)$ and t_i have already been defined above. When these experiments have been performed using SoftHASTE, it was not yet exactly clear how to implement the CC application to the HAGEN chip via the embedding FPGA. One idea is to use a real-time generator in hardware implementing a simple instruction that tells how to react to an incoming spike. An exponential decay seems to be a task that can be easily realized on an FPGA, it is therefore considered to be a good candidate. A concurring concept are CC lookup tables. In the end they were preferred, at least for the time being. Main reason for this decision was the manifoldness of shapes that could easily be applied by a simple software interface delivering arbitrary CC shapes. But an exponential conductance course has some additional advantages. A low temporal resolution is desired for time performance reasons. When being mapped to a coarse temporal resolution, a sharp peak is more likely to lose shape characteristics than a monotonic decay. Furthermore, two CCs generated at the same synapse shortly one after another can possibly not be superposed due to hardware limitations. The latter restriction is represented by eq. 3.3. In such a scenario the immediate onset of an exponential shape after an occurring spike might reduce the loss of information, compared with a time-to-peak larger than zero for conductance courses inspired by Shelley.

Mapping method for exponential shaping

It has not been clear from the start that the mapping approach from continuous to discrete values found for shapes described by eq. 3.1 would work satisfactorily with an exponential time course as well. So at first, either the CCI method for shape mapping was to be proven sensible, or another method had to be found. The basic setup of neuron bombardment was used as before. As it was expected to deliver pleasing results and thus further experiments possibly could be skipped, only CCI was applied for the beginning. The time constant τ in eq. 3.10 was set to be 2.5 ms, all other parameters were chosen just like for all previous experiments.

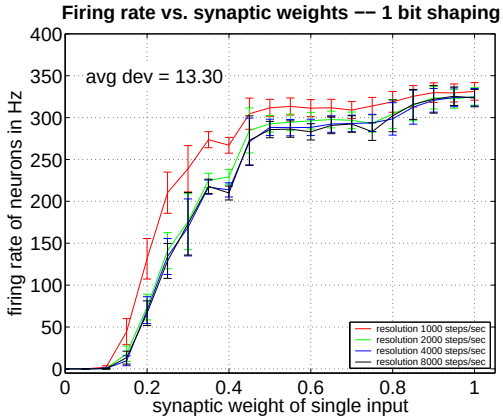
The result is plotted in fig. 3.10. Again the average standard deviation of the curves (as already defined for the matching experiments above) is included. In fact the conformity of curves obtained with different temporal resolutions for a given number of bits per synapse is at least as good as for CCs according to eq. 3.1.

Fig. 3.11 shows output firing rates for different bit resolutions for $\rho_{\text{temp}} = 4000\,\text{steps/sec}$, the mapping method was CCI. The result is satisfying, too.



(a) 4-bit shapes

(b) 2-bit shapes



(c) 1-bit shapes

Figure 3.10: Mean output firing rates of different setups using *exponentially decaying* conductance courses (see eq. 3.10) vs. synaptic input weight. The method mapping from continuous CC shapes to low temporal and amplitude resolution was CCI. The curves correspond to those in fig. 3.7 and show a pleasing matching.

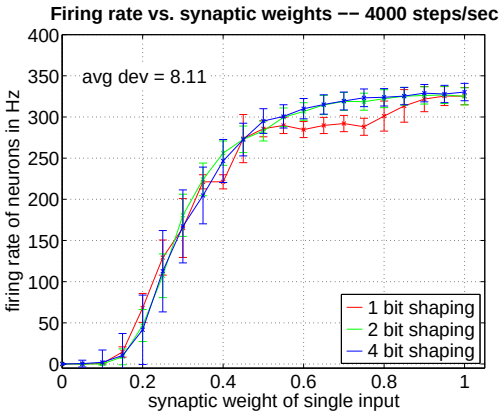


Figure 3.11: Output firing rate vs. input weight of synaptic inputs in single neuron bombardment. Standard setup and *exponentially decaying* conductance courses (see eq. 3.10) modeled with 1, 2 and 4 bits. The method mapping from continuous CC shapes to low temporal and amplitude resolution was CCI. The figure corresponds to fig. A.2(f), where Shelley CCs were applied.

Matching different shapes

As so much effort has been made to keep results of different bit resolutions comparable, it either might be useful for later experiments to find time constants for both Shelley's and Maass' CCs that allow a direct comparison between data obtained with each of them. Indeed, both shaping strategies have their advantages and form the basic shape library for all experiments presented in this study. Shelley suggested values for τ which result in a time-to-peak of $t_{peak} = 3\,\mathrm{ms}$ for CCs induced by excitatory synapses and 5 ms for the inhibitory case [30]. For a spike i arriving at time $t_i = 0$, eq. 3.1 can be written as

$$G^{\mathrm{Shelley}}(t) = c_{\mathrm{Shelley}} \frac{t^5}{\tau^6}\, exp\,(-t/\tau)\,\Theta(t) \tag{3.11}$$

with c_{Shelley} being an arbitrary constant. For $t > 0$ differentiation leads to

$$\dot{G}^{\mathrm{Shelley}}(t) = c_{\mathrm{Shelley}} \frac{t^4}{\tau^6}\, exp\,(-t/\tau) \left(5 - \frac{t}{\tau}\right) \qquad (t > 0) \qquad . \tag{3.12}$$

Thus Shelley's peak condition is solved by $\tau_{shelley} = \frac{1}{5}\, t_{peak} = 0.6\,\mathrm{ms}$ (1.0 ms) for excitatory (inhibitory) synapses. Within the HASTE setup all conductance courses are mapped to low resolutions according to fig. 3.3. Their amplitude is scaled in a way that the CC's peak always equals the maximum integer V_{max} that is codable by the available number of bits used per synapse.

For a scaled Shelley conductance course $\overline{G}^{\mathrm{Shelley}}(t)$, this means

$$\begin{aligned} \overline{G}^{\mathrm{Shelley}}(t_{peak}) = \overline{c}_{\mathrm{Shelley}}(\tau) \frac{(5\tau)^5}{\tau^6}\, exp\,(-5\tau/\tau) = \left(\frac{5}{e}\right)^5 \tau \quad &\stackrel{!}{=} \quad V_{max} \\ \iff \quad \overline{c}_{\mathrm{Shelley}}(\tau) \quad &\stackrel{!}{=} \quad V_{max}\, \tau \left(\frac{e}{5}\right)^5 \end{aligned} \tag{3.13}$$

In contrast to the original shape defined by eq. 3.11 which always encloses the same area underneath its curve, this amplitude manipulation generates a dependence of the course integral on the time constant τ. The integral over eq. 3.11 is

$$\int_{-\infty}^{\infty} G^{\mathrm{Shelley}}(t)\,\mathrm{d}t = \frac{c_{\mathrm{Shelley}}}{\tau^6} \int_{0}^{\infty} 5!\,\tau^5 e^{-t/\tau}\,\mathrm{d}t = 120\, c_{\mathrm{Shelley}} \qquad . \tag{3.14}$$

Including condition 3.13 eq. 3.14 results in a total shape integral for a scaled conductance course of

$$\int_{-\infty}^{\infty} \overline{G}^{\mathrm{Shelley}}(t)\,\mathrm{d}t = 120\, V_{max}\, \tau \left(\frac{e}{5}\right)^5 \equiv A^{\mathrm{Shelley}}(\tau) \qquad . \tag{3.15}$$

The value according to $A^{\mathrm{Shelley}}(\tau)$ for the case of exponentially decaying conductance courses (the superscript will be *Maass* as this shape was suggested by W.Maass in [15]) is much easier to calculate and finally leads to

$$\int_{-\infty}^{\infty} \overline{G}^{\mathrm{Maass}}(t)\,\mathrm{d}t = V_{max}\, \tau \equiv A^{\mathrm{Maass}}(\tau) \qquad . \tag{3.16}$$

The preceeding experiments indicate that constant integrals are an important criterion for the independence of output firing rates on conductance course variations. Thus the idea of assimilating the integrals of the shapes defined by eq. 3.1 and 3.10 in order to match output firing rates suggests itself. The condition to fulfill $A^{\text{Shelley}}(\tau_{\text{Shelley}}) \stackrel{!}{=} A^{\text{Maass}}(\tau_{\text{Maass}})$ then becomes

$$\frac{\tau_{\text{Maass}}}{\tau_{\text{Shelley}}} \stackrel{!}{=} \frac{A^{\text{Shelley}}(t)}{A^{\text{Maass}}(t)} = 120 \left(\frac{e}{5}\right)^5 \approx 5.70 \quad . \tag{3.17}$$

Regarding this, the corresponding value to $\tau_{\text{Shelley}} = 0.60\,\text{ms}$ is $\tau_{\text{Maass}} \approx 3.42\,\text{ms}$.

In case of short intervals between spikes running into the same synapse the resulting conductance changes are superposed if they are temporarily overlapping (at least this is what SoftHASTE does). As binary inputs are a very limited resource, the shapes applied to the synapses were scaled in a way that they always use the whole available integer spectrum. This means that the superposition has an upper limit, namely the value of a conductance course peak. When the sum of synaptic shapes succeeds this limit the maximum value is applied instead. In that case, information and input intensity is lost.

In order to avoid this falsifying effect, which probably would affect the Shelley shape with its later onset in a more critical way, the mean firing rate per input was reduced to 10 Hz in the experiment to be presented next. Once again, a single neuron was bombarded. To retain the whole possible output firing rate spectrum in the presence of low input rates, the neuron's threshold was set to -0.1 instead of the hitherto used values of -1. Resolutions of 4 bits for the conductance course amplitude and 8000 steps/sec in the time domain were selected to maintain resemblance to the original shapes. Both a bombardment experiment applying Shelley CCs with $\tau = 0.6\,\text{ms}$ and another one employing Maass CCs and $\tau = 3.42\,\text{ms}$ were performed. Fig. 3.12 shows the two shapes generated by HASTE for these settings.

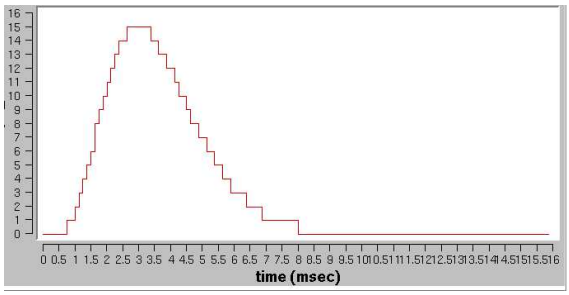
The results of the bombardment experiment are depicted in fig. 3.13. They confirm the assumptions made above.

The two curves match nearly perfectly for firing rates up to 75% of f_{max}, for higher rates they diverge slightly. This can be explained as follows: High output rates correspond to high input weights, which means that less active inputs are necessary to make the bombarded neuron fire. But for a small number of shapes being superposed within the neuron, the width of the courses becomes increasingly important. Having one more look at fig. 3.12 reveals that the exponential decay has a time above zero much longer than the Shelley CC and thus might support an overlay of different shapes. Hence, the area under a synaptic shape is *not* the only criterion for matching firing rates, but for large statistics it may be a sufficient one.

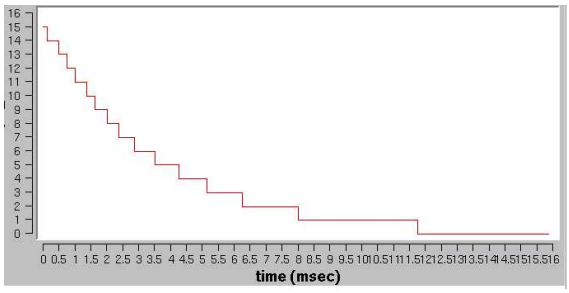
Conclusions drawn from this section

It has been found that a HASTE neuron with synaptic shaping according to Shelley and under biologically realistic input produces sensible output firing rates $\bar{f}_{out}$. The dependence of $\bar{f}_{out}$ on its input weights is clearly monotonic. Arithmetic rounding turned out to be not the optimal way of discretizing conductance courses. For 1-bit synapses and arithmetic rounding the dependence of $\bar{f}_{out}$ on temporal resolution can hardly be controlled. Keeping the temporal resolution fixed, the method matching $\bar{f}_{out}$ for different amplitude resolutions best was CCI. The latter caused $\bar{f}_{out}$ to respond nearly exactly the same for different temporal resolutions, too. Thus CCI will be applied in the experiments to follow.

A model describing the course of $\bar{f}_{out}$ for 1-bit synapses was developed. It reproduces the characteristics of the original qualitatively and after some technically inspired variations even



(a) Shelley CC



(b) Maass CC

Figure 3.12: Conductance courses as they are generated by HASTE for an amplitude resolution of 4 bit and a temporal resolution of 8000 steps/sec. Basis for (a) was a shape according to eq. 3.1 (Shelley), for (b) it was eq. 3.10 (Maass).

quantitatively.

As a second CC shape a plain exponential decay was introduced. The CCI method turned out to work well for it, too. The courses of $\bar{f}_{out}$ for both shaping methods were tried to be matched. This was achieved by adjusting the time constant of one of them on purely theoretical grounds. An experiment proved these considerations to be correct for low firing rates.

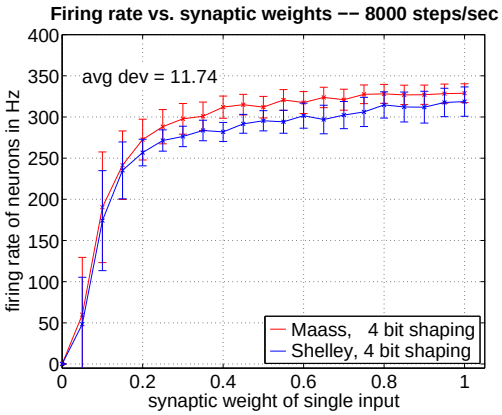


Figure 3.13: Output firing rate vs. input weight of synaptic inputs in single neuron bombardment. The setup is described and motivated in the text. The red curve denotes the output firing rate of a bombarded neuron receiving *exponentially decaying* CCs, the blue one CCs *according to Shelley*.

3.1.3 Interspike Interval Histograms

A common method in neuron research is recording time intervals between two subsequent spikes of a neuron's output. These are called interspike intervals (ISIs) and hold a lot of information about cellular dynamics and the neuron's input.

What ISIs can tell

Let $\hat{t}$ be the time the last output spike occurred on a neuron's axon. Then the probability density for the next spike to occur at time t is denoted as $P_I(t|\hat{t})$, the probability of a spike to occur in the interval $[t_1, t_2]$ is $\int_{t_1}^{t_2} P_I(t|\hat{t})\,\mathrm{d}t$, provided that $\hat{t} < t_1$. The lower index I represents the fact that $P_I(t|\hat{t})$ depends on the neuron input $I(t)$. The so-called *renewal theory* provides basic formalisms to handle the information given by such ISI recordings ([3], p.152-158). Two more quantities are introduced to statistically describe the firing behavior of a neuron. The *survivor function* $S_I(t|\hat{t})$ is defined by

$$S_I(t|\hat{t}) = 1 - \int_{\hat{t}}^{t} P_I(t'|\hat{t})\,\mathrm{d}t \quad . \tag{3.18}$$

It is the probability for the neuron not to fire (to "survive") between $\hat{t}$ and t. The second quantity, $\rho_I(t|\hat{t})$, is called *hazard function* and describes the rate of decay of $S_I(t|\hat{t})$. It is therefore a measure for the neuron's actual output spike rate at time t.

$$\rho_I(t|\hat{t}) = \frac{-\frac{\mathrm{d}}{\mathrm{d}t} S_I(t|\hat{t})}{S_I(t|\hat{t})} \quad . \tag{3.19}$$

All three quantities describing the statistics of a renewal process can be transformed into each other. $P_I(t|\hat{t})$ is the easiest one to be measured experimentally, as a histogram of the ISIs in a sufficiently long spike train is considered to be a good estimate for it.

In what follows t_{ref} denotes the absolute refractory period of a neuron. A simple calculation (see [3]) shows that if a hazard function is given such as

$$\rho_I(t|\hat{t}) = \begin{cases} 0 & \text{for } t \le \hat{t} + t_{\mathrm{ref}} \\ c & \text{for } t > \hat{t} + t_{\mathrm{ref}} \end{cases} \quad , \tag{3.20}$$

the probability distribution is an exponential decay starting at $t = t_{\mathrm{ref}}$

$$P_I(t|\hat{t}) = \begin{cases} 0 & \text{for } t \le \hat{t} + t_{\mathrm{ref}} \\ A\, e^{-t/\tau} & \text{for } t > \hat{t} + t_{\mathrm{ref}} \end{cases} \quad . \tag{3.21}$$

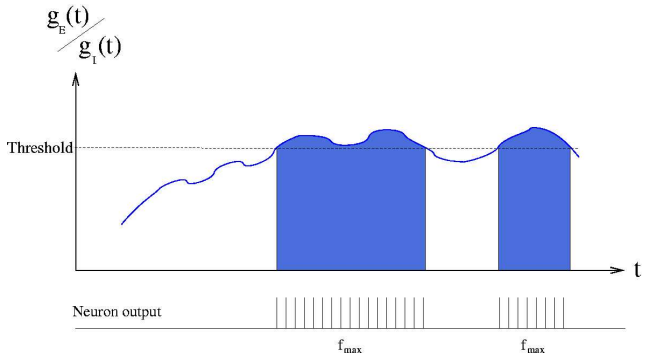
Here, c and A are constant positive values. This example will be relevant for the following experiments since Poisson distributed inputs are used to activate the neuron's membrane potential.

The neural dynamics of HASTE respectively HAGEN is quite simple and well understood since it is a full custom design. So why make ISI recordings of a HASTE neuron's output? Assume that the neuron is under a biologically realistic input bombardment. Different ways to model the conductance courses superposed within the neuron may lead to different ISI histograms. For example long drawn-out shapes may smoothen the temporal effect of a single input spike, i.e. make the neuron less sensitive for the exact arrival time of incoming spikes. Applying such CCs to the input synapses of a HASTE neuron might facilitate interactions

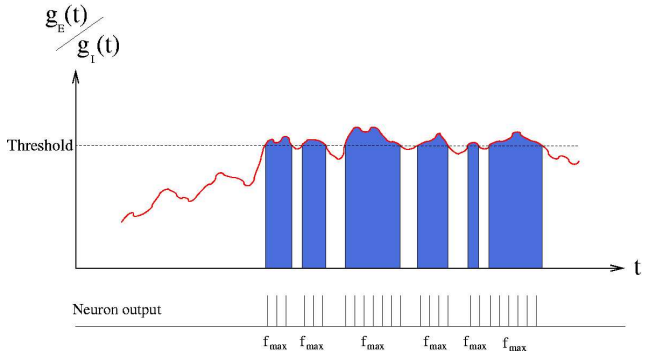
between two non contemporaneous spikes, but affects the membrane potential like a low pass filter (see fig. 3.14 for illustration).

As long as a HASTE neuron's cortically induced conductance ratio is above threshold it constantly fires at its maximum rate. This rate is given by $f_{max} = \frac{1}{t_{\text{ref}}}$. Thus the frequency of threshold crossings directly affects the ISI distribution. See fig. 3.14 for illustration. A conductance ratio changing slowly will exhibit longer periods above threshold compared to a very input sensitive regime with high frequency fluctuations, assuming the average ratio being close to threshold. Hence an inertly changing conductance ratio results in the dominance of the shortest ISI possible.

The *all-or-nothing* spiking behavior of HASTE is not very realistic, especially the implemented absolute refractory mechanism is too simple. But this deficit may be acceptable, if the conductance ratio fluctuates around threshold with a frequency high enough to obtain a broad ISI spectrum, which is a necessary condition for temporal coding.



(a)

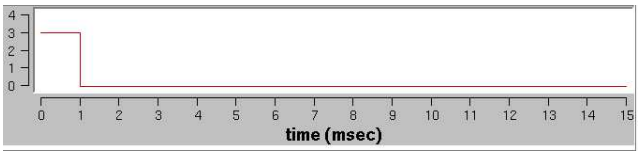


(b)

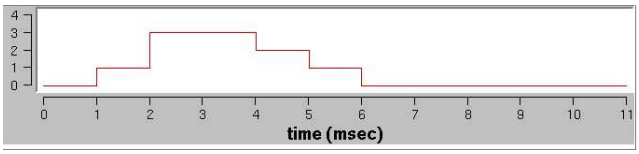
Figure 3.14: (a) and (b): Conductance ratio $g_E(t)/g_I(t)$ of a neuron exposed to many synaptic inputs (schematic). The curve is the sum of all incoming CCs. Below the curve the neuron's output is shown. For (a) the shapes generated at the synapses are long and smooth, for (b) they are short and sharp. The periods of the ratio being above threshold are colored blue. During these times a HASTE neuron fires at its maximum rate f_{max}, all the remaining time the rate is zero.

The experiments described here have been carried out to find out if the behavior of HASTE in terms of interspike intervals corresponds to biological scales. A major deficit of the HAGEN chip, namely its small number of synapses, was circumvented by setting the firing rates of all inputs higher than typical values in nature. Hence, one HASTE input could be said to represent many biological ones.

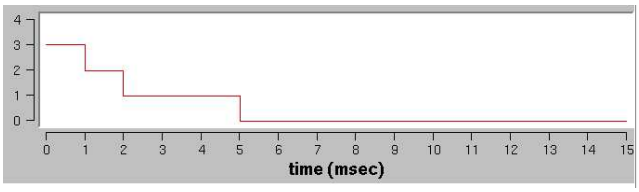
An ISI histogram recorded from a real neuron in human hippocampus can be seen in



(a) Single Cycle CC ("pseudo delta peak")



(b) Shelley CC



(c) Maass CC

Figure 3.15: Three different shapes used to simulate conductance courses for the experiments described in sec. 3.1.3. (a) is the shortest CC possible, i.e. maximum value for one time step, others zero. (b) shows a shape defined by eq. 3.1, with $\tau = 0.6\,\text{ms}$. (c) represents an exponential decrease according to eq. 3.10 with a time constant of $\tau = 2.5\,\text{ms}$.

fig. 3.16. The upper plot, measured in awake state (High Conductance States usually occur in the awake brain), will be taken as a landmark when judging the histograms produced by HASTE.

ISI recordings with HASTE

Just like in the first section of this chapter a single neuron was bombarded with 30 Poisson spike trains, the weight was the same for all inputs and it was gradually increased from 0.0 hwu to 1.0 hwu. Each of the trains had a mean firing rate of 33 Hz. The absolute refractory period was set to 3 ms. A bias input weighted with -1.0 hwu adjusted the threshold. The conductance course resolution ρ_{amp} was set to two bits. Three different CC shapes were applied: A first run was done with CCs lasting for exactly one network cycle, which is the shortest shape that larger than zero a CC generator can produce and represents a *pseudo delta peak*. Another shape was generated according to Shelley [30], see eq. 3.1, with $\tau = 0.6\,\text{ms}$. A third one was modeled according to eq. 3.10, with $\tau = 2.5\,\text{ms}$. This shape was suggested, among others, by Wolfgang Maass [16]. See fig. 3.15 for illustration.

Experimental results are shown in figures 3.17, 3.18 and 3.19. For each of the three CC modeling methods the histograms for three different input weights are plotted.

The probability that the conductance ratio exceeds the threshold grows for higher input weights, thus the ISI distribution shifts to smaller values. The absolute refractory period of 3 ms set can be seen as a gap of no events at the left of each histogram. The ISI distribution

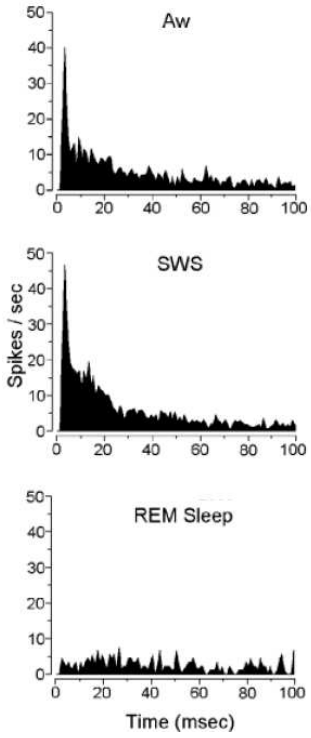


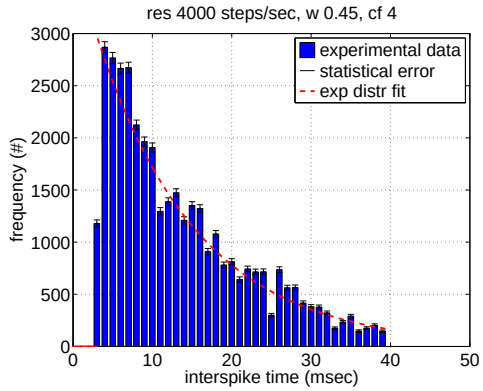
Figure 3.16: A real interspike interval histogram taken from a publication of Staba et al [32]. It shows the ISI distribution of a single neuron in human hippocampus during awake state (Aw), slow wave sleep (SWS) and REM sleep.

belonging to the single-cycle CCs (fig. 3.17) can be very well matched by an exponential fit. This is easily explained by the considerations already made above (see eq. 3.20 and 3.21). The occurrence of an output spike is completely independent from earlier spikes or others to come, because the CCs do not last for longer than a single network cycle. That's why the output spiking probability must be Poisson distributed just like the input spiking probability and therefore results in an exponential ISI distribution.

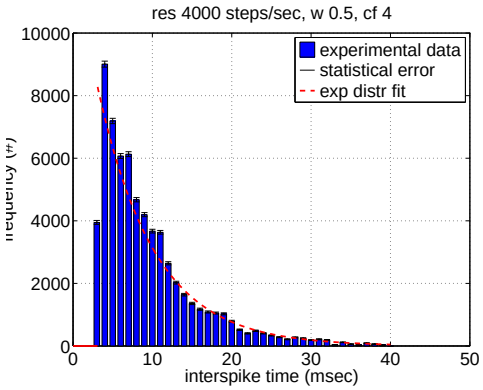
In case of long drawn-out CCs there often *is* a correlation between two subsequent output spikes. Hence the Poisson distribution of the output spike times respectively the exponential character of the ISI probability density is distorted. Just as predicted the long drawn-out CCs a la Shelley and Maass led to a high dominance of the shortest ISI possible.

Conclusions drawn from this section

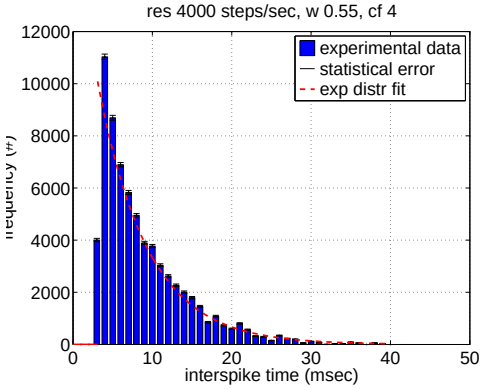
A HASTE neuron under biologically realistic input can be configured in a way that its interspike intervals are distributed to a spectrum broad enough to ensure the possibility of temporal coding. Prolonging synaptic responses to an incoming spike results in growing dominance of the shortest interspike-time possible. By balancing the ratio of input weight to threshold a broad spectrum of interspike-times can be established, though. An in-vivo measurement of a real neuron's interspike intervals exhibits a spectrum width in the order of 40 ms. The simulation experiments performed on HASTE resulted in spectrum width values of about 10 to 40 ms, depending on synaptic shaping and the ratio of input weights to threshold. In spite of its plain refractory mechanism the HASTE system obviously can uphold interspike diversity if it is exposed to biologically realistic activity.



(a)

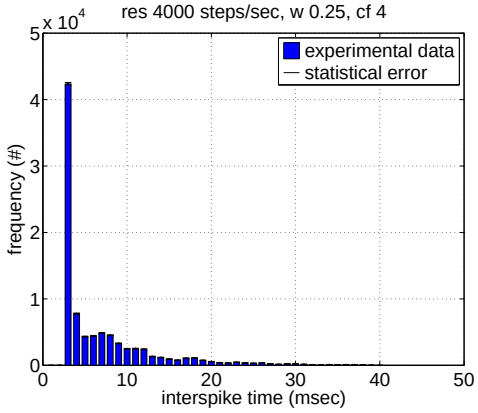


(b)

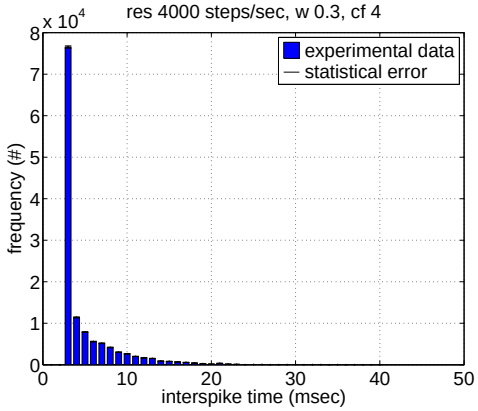


(c)

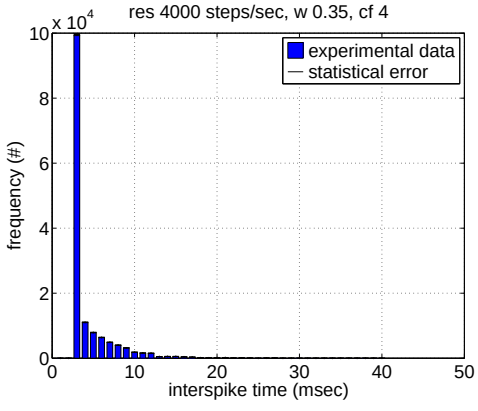
Figure 3.17: Interspike Interval Histograms recorded from a HASTE neuron bombarded with 30 Poisson distributed spike trains. The synapses at the neuron's input applied single cycle CCs (fig. 3.15(a)). Temporal resolution was 4000 steps/sec for the experiment and is 1 ms/*bin* for this histogram. The bin most left includes interspike times smaller than the refractory time and therefore holds less counts than his right neighbor. From (a) to (c) the synaptic weight of the inputs is increased.



(a)

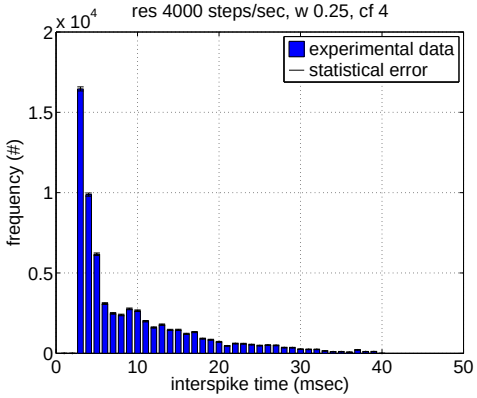


(b)

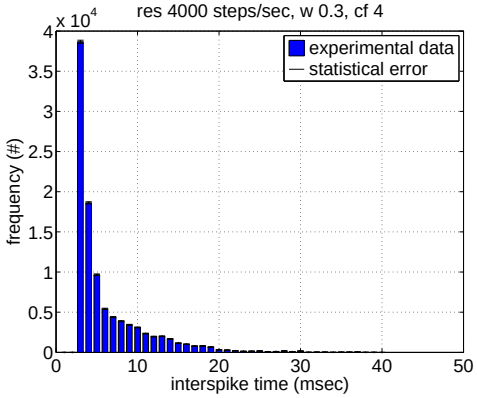


(c)

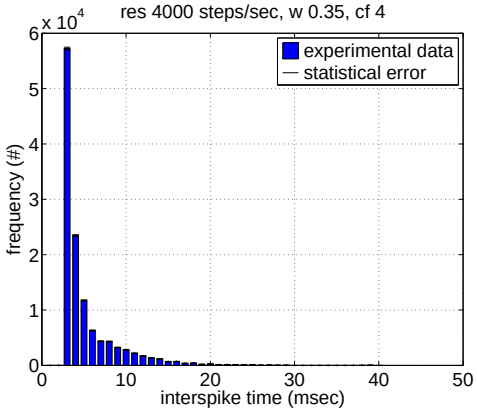
Figure 3.18: Interspike Interval Histograms recorded from a HASTE neuron bombarded with 30 Poisson distributed spike trains. The synapses at the neuron's input applied CCs according to eq. 3.1 (fig. 3.15(b)). Temporal resolution was 4000 steps/sec for the experiment and is 1 ms/*bin* for this histogram. From (a) to (c) the synaptic weight of the inputs is increased.



(a)



(b)



(c)

Figure 3.19: Interspike Interval Histograms recorded from a HASTE neuron bombarded with 30 Poisson distributed spike trains. The synapses at the neuron's inputs applied CCs according to eq. 3.10, i.e. plain exponential decays (fig. 3.15(c)). Temporal resolution was 4000 steps/sec for the experiment and is 1 ms/*bin* for this histogram. The bin belonging to the leftmost bar includes both interspike times smaller and larger than the refractory period. It therfore holds less counts than its right neighbor. From (a) to (c) the synaptic weight of the inputs is increased.

3.2 Liquid Computing with HASTE

Now that HASTE was "put to the acid" and proved useful, more elaborate experiments were conceivable. Basic concepts of *liquid computing* and the approach of generating liquids from binary neural networks were introduced in sec. 1.3. A Perceptron implementation developed in the working group of this study's author was mentioned also. Technical details about adaption to perform liquid computing on the HASTE system have been proposed in sec. 2.1.

On the one hand liquid computing on a VLSI Perceptron already had been performed within the *Electronic Vision(s)* group and thus some necessary knowledge and methods were easy to obtain [28]. On the other hand Maass et al. have successfully applied liquid computing to biologically oriented networks of integrate-and-fire neurons in a pseudo-continuous time domain [15]. HASTE was considered to be a candidate to bridge this gap. It implicitly contains the ability to work as a plain Perceptron but can be improved towards biologically realistic behavior.

To begin with, one of the basic experiments presented in [28] was reproduced on the HASTE system as a proof of principle. Afterwards variations allowed by the HASTE system have been applied to the setup in order to gain improvements in terms of memory capacity. The idea behind this was that long drawn-out synaptic shaping may help the liquid to store information not within the output states of the neurons and the mechanism of feedback, but already at the synapses respectively the inputs. This approach was carried out in a stepwise way, starting from the basic setup according to [28] and moving towards more sophisticated synaptic shaping and timing.

Advancing the search for memory capacity optimization the networks suggested by Bertschinger et al. [1] then were left behind and topologies inspired by Wolfgang Maass' work on liquid computing with leaky integrate-and-fire neurons have been applied [15].

3.2.1 The "Edge of Chaos" with HASTE

The "edge of chaos" proposed by Bertschinger [1] and reproduced on the HAGEN chip by the author of [28] denotes a band in the two-dimensional parameter space spanned by k and σ^2. These two parameters determine the generation of networks which are utilized as liquids, see sec. 1.3.2. The liquids are part of LSMs trained to approximate functions on a binary input bit stream connected to every neuron within the net. The edge of chaos forms a border of qualitatively different regions in terms of the liquids' activity. Along this border, respectively on the edge of chaos, the liquids memory capacity was found to peak [1].

To find this edge, a parameter sweep covering a sufficiently large region is necessary. Nearly all experiments to follow will basically be such two-dimensional samplings of the space spanned by k and σ^2.

P-HASTE

In previous sections the possibility of writing just a pseudo delta peak to the synaptic lookup tables has already been used. This means that in case of an incoming spike a synapse applies maximum input to its target neuron for exactly one network cycle and afterwards is quiet again. If all binary inputs forming a multi-bit synapse are active, this maximum input always affects the target neuron with exactly the same total weight independent of the number of bits used for shaping (see sec. 2.1).

When applying these pseudo delta peaks and setting both the refractory time of all neurons and the transmission delay of all synapses to zero, the setup should just make HASTE work like the HAGEN chip without any extensions. The only exception made for the experiments to follow was the usage of the FPGA based feedback router providing a feedback more flexible than that of HAGEN alone. Nonetheless this definite configuration should make HASTE behave like a plain VLSI Perceptron and on that score it will be called *Perceptron* HASTE or P-HASTE. In order to verify this claim an experiment designed to be run on HAGEN was repeated on P-HASTE, see subsection "First success".

Basic setup

Big parts of the software utilized for the experiments described here were adopted without changes from Felix Schürmann, the author of [28]. Some of them had to be adapted to the requirements of multi-bit shaping. For all liquid computing experiments HASTE was utilized to implement a network representing the liquid. If not explicitly stated differently, the neurons were connected and driven by a Bertschinger-like input bit stream, according to the instructions given in [1] and explained in sec. 1.3.2. The memory capacity was sampled for every point on a grid covering the parameter space spanned by k and σ^2. For each sampled point the memory capacity was evaluated by training a linear readout connected to the network to solve the three-bit parity task described in sec. 1.3.3.

The MC of a liquid state machine is defined as the sum over the mutual information in this system for all time-shifts $\tau \leq 0$ (see sec. 1.3.3). For a distinct τ training and evaluation of the linear readout was performed as follows:

For two times 2000 cycles a randomly generated input bit stream was applied to every neuron in the network and the resulting liquid states were recorded. The target output according to the demanded three-bit parity problem was calculated for each time step. Using these target values and the liquid states belonging to the first 2000 input cycles, the linear readout then was calculated as proposed in [28]. After this training phase the readout was kept fixed and its three-bit parity prediction for the liquid states referring to the next 2000 cycles was computed. Finally the mutual information included in the system was calculated according to eq. 1.17.

As it was not possible to evaluate all $\tau \leq 0$ in practice, only a finite number of evaluations were performed. If τ was larger than $\tau_{\max}$ or if the mutual information dropped below a lower limit $\mathrm{MI}_{\min}$, evaluation was stopped. In [28] the monotonic and fast dropping of mutual information for increasing values of τ can be seen, so this necessary truncation was not expected to be source of a significant error.

The memory capacity was calculated not only once for each data point but multiple times in order to obtain information about the reproducibility of a certain parameter set's performance. For each repetition a new network was generated while the applied input bit stream was reused. Evaluating this statistical information, a mean value and a standard deviation was found for the MC at each sample point in the regarded region. Since the parameter σ^2 denotes the variance of randomly chosen synaptic weights and the HAGEN weights are limited to a bounded region $[-1, 1]$, values of $|\sigma^2| > 1$ make no sense. The maximum number of inputs that can be fed back into one neuron is limited to a value $k_{\max}$ due to the finite number of input synapses per neuron. For a HASTE neuron $k_{\max}$ is given by eq. 2.3. Therefore the region to be scanned could be cut down to a finite area.

Successful reproduction of a liquid computing experiment

The first goal was to reproduce basic results presented in [28] utilizing the new platform, i.e. to find the edge of chaos using the HASTE system instead of a plain HAGEN. On P-HASTE a parameter sweep with k running from 1 to 16 in steps of 1 and σ^2 running from 0.0 to 1.0 in steps of 0.05 was performed. 256 neurons were used, although only the outputs of 240 could be used for feedback (see sec. 2.1). The remaining 16 neurons received k inputs and thus were able to improve the memory capacity, but they were not fed back. Regarding only the 240 feedback-capable neurons the isotropy of the network generation is as good as possible because the flexible FPGA routing was used. Each of those neurons could be fed back to every other neuron within the net. This is an improvement compared to the topologies applied in [28] since perfectly following Bertschinger's instructions was not possible on a stand-alone HAGEN due to limited preset feedback wires.

The input proposed by Bertschinger was a single stream consisting of values $\in \{\bar{u}-1, \bar{u}+1\}$ connected to every neuron. Regarding the experience gained in the *Electronic Vision(s)* group $\bar{u}$ was a priori set to zero. This always exhibited the best results in terms of memory capacity. Since a HASTE neuron's input only can be 0 or 1, the Bertschinger-like input stream consisting of only one signal was realized as two mutually exclusive signals, one of them representing the inverse of the other. One was connected to every neuron with a weight of 0.5, the other with -0.5. This possible difference in affecting target neurons of value $(0.5-(-0.5)) = 1$ accounts for a neuron's maximum output change of $(1-0) = 1$.

The parameters for stopping the MC evaluation were set to $\tau_{\max} = 10$ and $\mathrm{MI}_{\min} = 0.001$. For all experiments shown here the stopping criterion always was MI falling below $\mathrm{MI}_{\min}$, so $\tau_{\max}$ was chosen large enough to not significantly lose information.

Fig. 3.20 shows the P-HASTE results for 256 neurons, the edge of chaos is an obvious band of high MC mean values. The positions of the first three measured MC maxima are marked (1.dot, 2.star, 3.cross), the values are $m_1 = 3.38$, $m_2 = 3.38$ and $m_3 = 3.35$. The corresponding standard deviations over all 30 runs are $s_1 = 0.24$, $s_2 = 0.20$ and $s_3 = 0.23$. The errors of the mean values can be estimated by $\Delta^{gen}_{\mathrm{m_i}} = s_i/\sqrt{N}$ with $N = 30$. For all three maxima $\Delta^{gen}_{\mathrm{m_i}} \approx 0.04$. This error represents the fluctuation caused by multiple network generation (hence the superscript "gen"), i.e. it is a measure for the reliability of a certain parameter set to deliver a powerful liquid. It does *not* contain errors resulting from a too sparse parameter sampling. Within the error range the MC values found with HASTE are the same as those presented in [28].

The proximity of the first three maximal values in the scope of their errors and their significant distance in parameter space indicates a band of roughly constant MC values along the bend of the edge, or pictorially spoken, a sort of crest in the "MC landscape".

In order to extend synaptic signals to more than one bit the number of neurons had to be reduced. Comparing results of different HASTE setups would only make sense if the number of used neurons was the same for all experiments. Thus, intending to work with two-bit conductance courses, the same experiment as the one shown in fig. 3.20 was performed with 120 neurons only, see fig. 3.21. Again the band of high computational power is visible, now the first three measured maxima being 2.60 ± 0.04, 2.60 ± 0.03 and 2.58 ± 0.05 (again the errors result from network generation). For input shaping *two* bits were selected because they provide the possibility of modeling input courses while still many neurons are available.

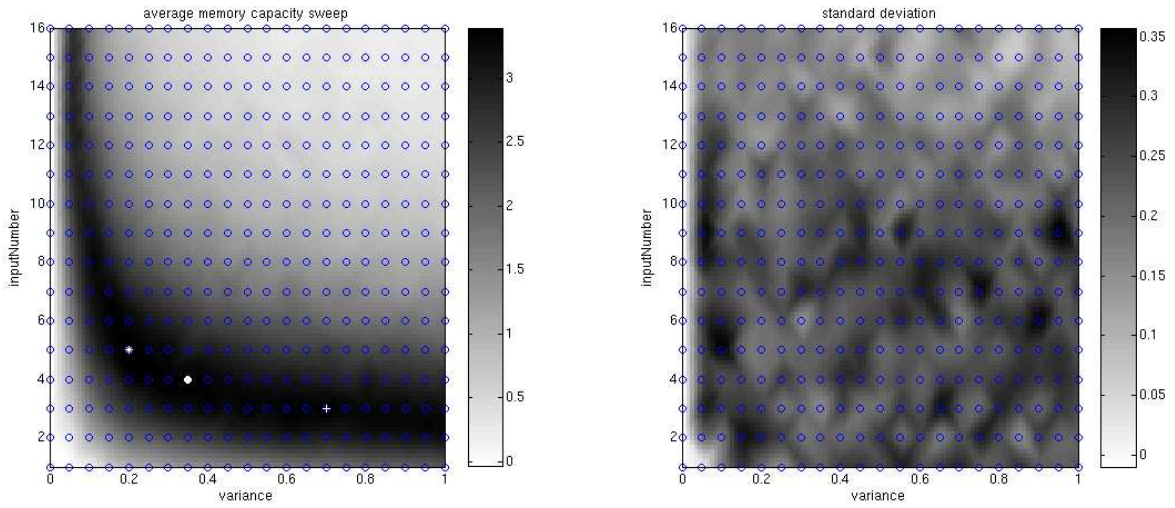


Figure 3.20: Left: Memory capacity (grey value) of liquids generated according to Bertschinger as a function of two parameters determining the network topology (input number and variance). Each data point represents the mean over 30 liquids generated with the same parameters and receiving the same input stream. Right: Each data point denotes the standard deviation of the MC measured at this point in the parameter space over 30 runs. The readout of the liquid state machine was trained to predict the parity of three consecutive input bits fed in at a distinct time earlier. The HASTE system was configured to work as a plain Perceptron, i.e. the synapses applied *pseudo delta peaks* responding to spikes, refractory time and synaptic transmission delay were set to zero. The network consisted of 256 neurons.

Chaos and order

The "edge of chaos" was claimed to divide two regions of characteristically different liquid activity in parameter space. This still has to be proven. Having a look at the output patterns acquired in two different points of parameter space, one from the left of the band and one from the right, the naked eye can recognize something like ordered and disordered activity (see appendix A.1, figures A.3 and A.4).

According to the definition, for a chaotic system different preconditions have to lead to significantly different states. A method to test this for the HAGEN liquid was developed by Felix Schürmann. In what follows an *input bit stream* always denotes a one-dimensional stream of bits with equal probabilities for 0 and 1. The test is presented step by step: For a distinct point in parameter space a network is generated. S_1, S_2 and S_{test} are three independently generated input bit streams, S_1 and S_2 having the same length. First S_1 is applied to the network, directly followed by S_{test}. Then the network state and all information stored in the synapses or feedback devices is reset. Afterwards S_2 is applied, again followed by S_{test}. During the application of S_{test} the output states of the network are recorded. The purpose of this procedure is to put two identical networks into completely different initial states and, with the beginning of a shared input stream S_{test}, observe whether they synchronize again, i.e. converge to a sequence of similar or identical states, or if they do not. Since the network

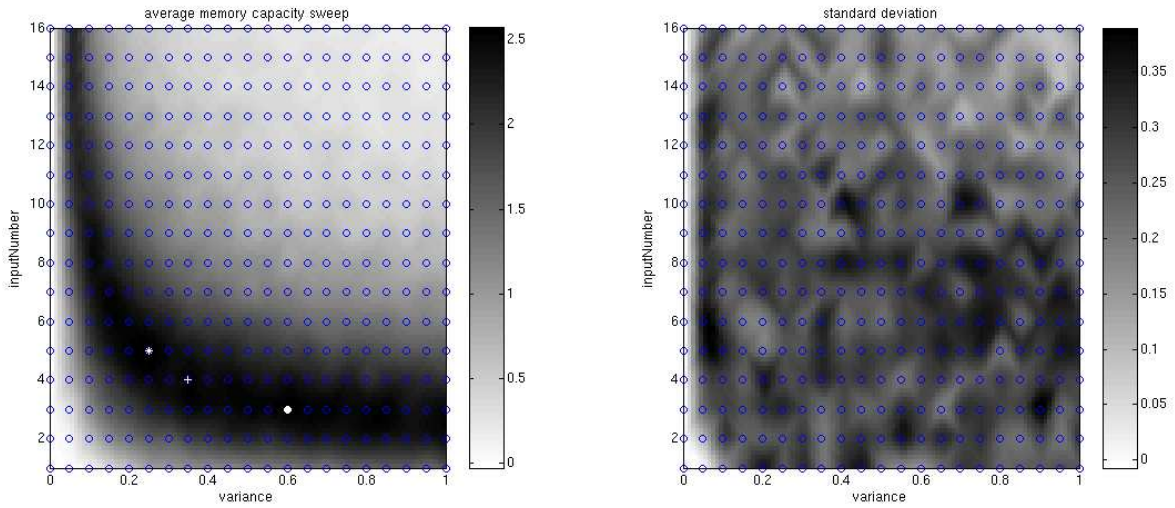


Figure 3.21: The same experiment as the one shown in fig. 3.20, but with 120 neurons only. Again mean MC values are plotted to the left, standard deviations to the right.

states are vectors of binary digits, the hamming distance was considered to be a good measure for their resemblance. Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two k-dimensional vectors of binary digits. Then the hamming distance is defined as

$$\mathrm{d_{ham}} = (\boldsymbol{x} - \boldsymbol{y})^2 \qquad , \tag{3.22}$$

i.e. if the states are identical, the distance becomes zero. In order to compare hamming distances obtained for vectors with different values for k they need to be normalized,

$$\mathrm{d_{ham,norm}} = \mathrm{d_{ham}/k} \qquad . \tag{3.23}$$

On average two randomly generated binary vectors with 0 and 1 equally distributed the mean normalized hamming distance is 0.25.

Be s^i_{test} the ith bit of S_{test} and $\boldsymbol{o}^i_1$ ($\boldsymbol{o}^i_2$) the corresponding output state during the first (second) application of S_{test}. The normalized hamming distance is evaluated for all measured pairs of $\boldsymbol{o}^i_1$ and $\boldsymbol{o}^i_2$. The course of $\mathrm{d_{liquid}(i)} \equiv \mathrm{d_{ham,norm}}(\boldsymbol{o}^{\mathrm{i}}_1, \boldsymbol{o}^{\mathrm{i}}_2)$ then can be plotted versus i.

Using Felix Schürmann's tool the P-HASTE experiment with 256 neurons was studied applying this method. The proposed procedure was applied to every point in the parameter space where the memory capacity already had been measured at. The result can be seen in fig. 3.22. The initialization phase, i.e. the application of S_1 respectively S_2, was performed for 1000 cycles each. This large number accounts for the fact that a network reset as postulated above was not possible for HASTE within one experiment due to technical reasons. This is a task still to solve. But, regarding a maximum memory capacity in the order of 4 for the evaluated networks, a complete loss of information about conditions 1000 cycles ago can be assumed. After the different initializations of both identical networks the course of $\mathrm{d_{liquid}(i)}$ was measured for 300 more cycles, applying the same input bit stream. For every pair of parameters the hamming distance evaluation was performed 30 times in order to average over fluctuations.

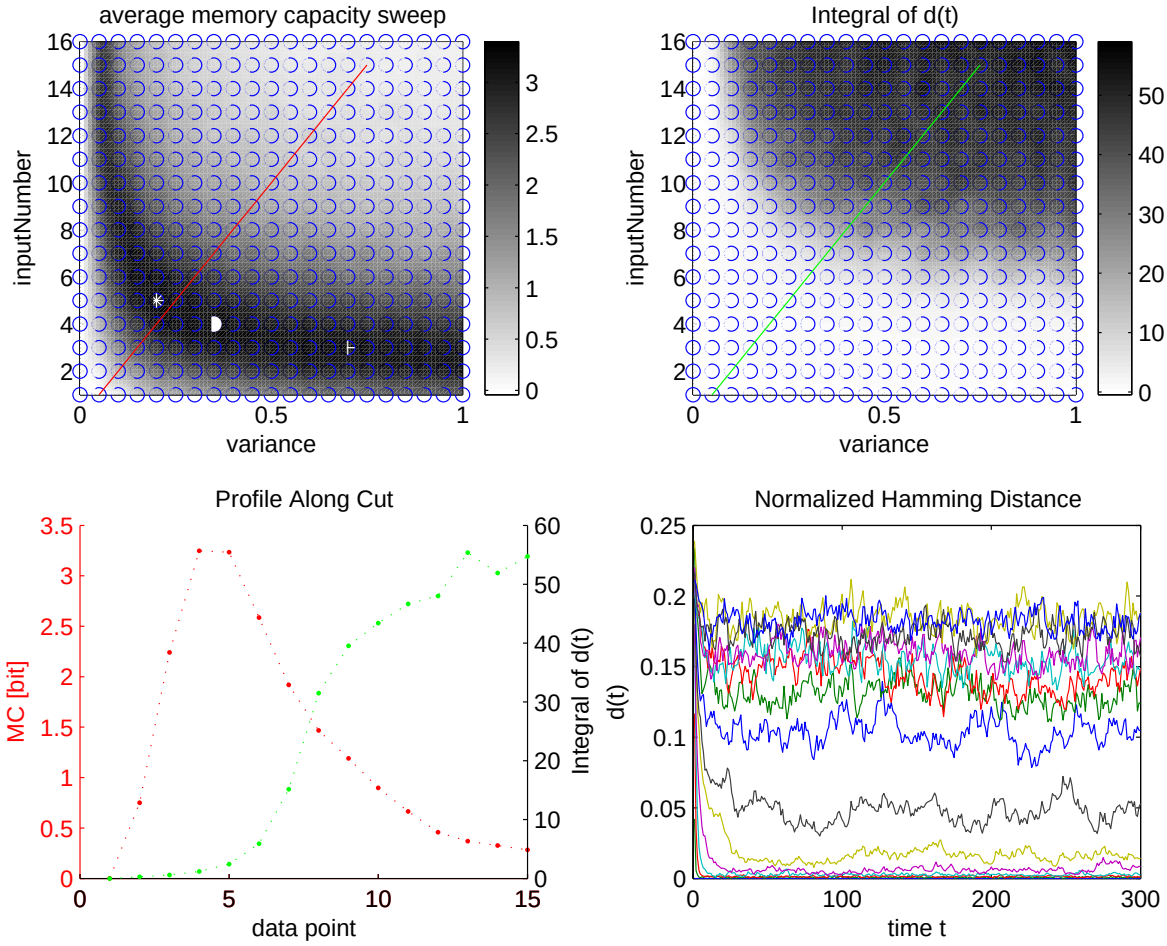


Figure 3.22: Upper left: Same as left part of fig. 3.20. For the parameter pairs on the red line the memory capacity is re-plotted in the lower left figure (red). In another experiment a single network was generated for every parameter point the memory capacity was evaluated at (blue circles). This network was fed with the same input bit stream twice, being differently initialized in each case. The hamming distance $d(t)$ between the generated output states was recorded. The lower right figure shows the course of this hamming distance for every parameter point which is crossed by the green line in the upper right plot. More precisely, every plot in the lower right figure represents the mean over 30 courses evaluated for a fixed parameter set. Lower courses of $d(t)$ correspond to lower values of k. Every data point in the upper right plot denotes the integral over such a mean course. The integrals of all parameter pairs crossed by the green line (same as for the red one) are re-plotted in the lower left figure, too.

The course of the average hamming distance $\overline{\mathrm{d}}_{\mathrm{ham,norm}}$ along the 300 cycles of synchronized input is shown for 15 selected parameter sets in the lower right subplot of fig. 3.22. Immediately after the onset of identical input streams, $\overline{\mathrm{d}}_{\mathrm{ham,norm}}$ is about 0.25 for all parameter sets. This was expected, see eq. 3.23. The differently initialized but identical networks then tend to synchronize, i.e. the hamming distance decreases. Some of them nearly completely adapt to each other, which corresponds to values of $\mathrm{d}_{\mathrm{ham,norm}} \approx 0$. Others seem to converge to a finite value $\mathrm{d}_{\mathrm{ham,norm}} \approx \tilde{\mathrm{d}} > 0$. This means that different preconditions lead to different final states, which was the criterion for chaos. Of course this cannot be proven to infinity, but since 300 cycles are a relatively long time compared to memory capacities of $\tilde{4}$, a characteristic difference between the networks can be claimed. The so-called "edge of chaos" really divides networks with ordered from those with chaotic activity. Parameters drawn from this edge generate networks of high memory capacity.

Obviously $\tilde{d}$ depends on the parameters the network has been generated with, $\tilde{d} = \tilde{d}(k, \sigma^2)$. The 15 selected parameter sets form a cross-section diagonally through the band of high memory capacity. In the lower left of fig. 3.22 both the memory capacity and the integral over the mean hamming distance after input synchronization are plotted for every point along the cross-section. The memory capacity (red) reaches its peak right when the hamming integrals (green) begin to leave the x-axis, i.e. at the onset of chaos. Similar measurements have been performed for cross-sections cutting the band of high memory capacity horizontally and vertically (see appendix A.1, figures A.5 and A.6), leading to the same conclusion.

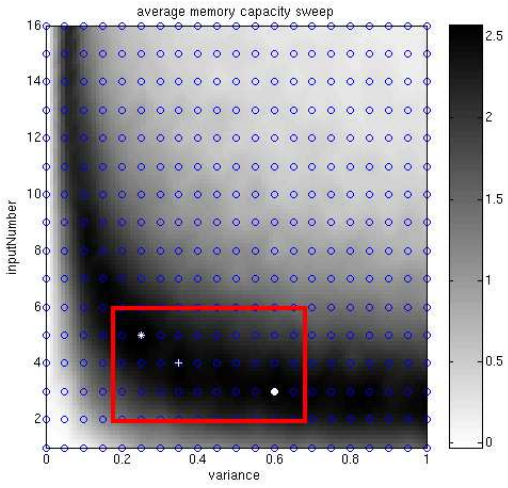
How to assess different HASTE setups

Assume the number of used neurons to be fixed. Then, apart from the network topology, the P-HASTE configuration is non-ambiguous. But having such a flexible extension like HASTE to one's disposal, a way suggesting itself is to leave P-HASTE behind, elaborate synaptic behavior and compare computational power of liquids with different setups (see sec. 3.2.2). This creates the need for a measure to somehow assess HASTE configurations regarding their capability to build substrates for good liquids. Even for P-HASTE one might be interested in the dependence of its liquid performance on a variable number of neurons. The best memory capacity over all possible network generation parameters that reproducibly can be achieved with a certain HASTE setup was regarded as such a measure.

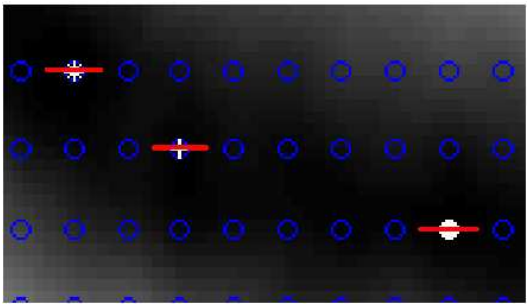
Due to the impossibility of continuously scanning a finite part of the two-dimensional parameter space, it is an unsolvable task to find the exact global maximum memory capacity. The problem has to be solved in a different way. Assume the "MC surface" over the parameter area of interest can be proven to be somewhat smooth, i.e. it is continuous and fluctuations can be estimated quantitatively. Then for each sampling resolution an approximation of the maximum memory capacity $\overline{\mathrm{MC}}$ within the scanned parameter space including an error estimation can be found. Hence the goal is firstly to show the memory capacity over the parameter space of interest to be continuous. Then an estimation has to be given for the MC fluctuation within the small area A_{sam} captured by one sample (= *sample patch*). Due to the fact that one of the two parameters suggested by Bertschinger is an integer that cannot be resolved more precisely, the parameter space has the form of a grating, A_{sam} being only a line in this case. But in later experiments two continuous parameters were swept, so the terminology will be kept general.

The degree of smoothness of the MC surface around the edge of chaos was proven as follows: Let S_i be the set of points in parameter space which technically can be resolved

within a sample patch $A_{\text{sam,i}}$. As already noted above, for the Bertschinger-like parameter sweeps these patches are just lines. To get a measure for MC fluctuations within $A_{\text{sam,i}}$, the MC for every element of S_i was measured in the same way as for the coarse sample grid. Fig. 3.23 shows the sample patches belonging to the first three measured MC maxima of the 120 neuron experiment.



(a) MC over parameter space



(b) High resolution scans

Figure 3.23: (a) re-shows the left plot of fig. 3.21, marking a smaller region that contains the first three MC maxima. (b) is a close-up of this region. For the coarse parameter space sampling each of the three MC maxima represents an area indicated red (in the text this is called sample patch). A parameter sweep with maximum resolution was performed along these lines to obtain the MC distribution (see fig. 3.24).

Now having a set of 51 MC values $F_i = \left\{mc_j^i,\, j = 1...51\right\}$ for 51 finely resolved and equidistant parameter points, a histogram can be plotted. The deviations from the mean over all 51 values represent the deviations of the MC surface over one A_{sam} from flatness, including the fluctuations caused by multiple network generation already discussed above. Fig. 3.24(a) shows the histogram of MC values found in the area A_{sam} around the first MC maximum of the P-HASTE experiment (120 neurons) before translation.

The data looks encouraging since the measured deviations can be well fitted by a normal distribution with $\sigma \approx 0.05$, which is only slightly larger than the included fluctuation of $\Delta^{gen} \approx 0.04$ caused by network generation (see subsection "Successful reproduction..."). The same measurements were performed for the second and third maximum, delivering normal-

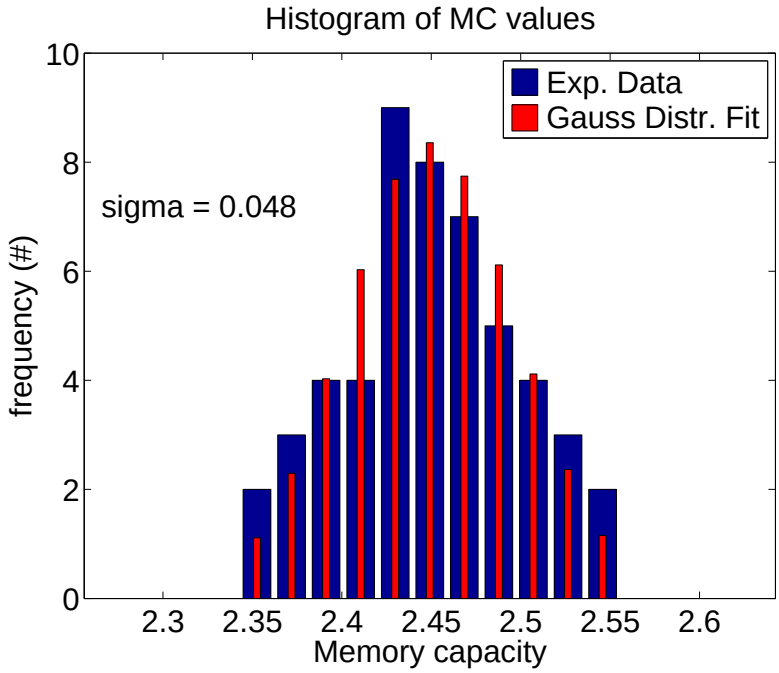


Figure 3.24: Histogram of 51 MC values (blue bars). The binned values belong to 51 different parameter sets determining the network structure according to Bertschinger et al. The (k, σ)-pairs were uniformly distributed around the first MC maximum found in the coarse parameter sweep of fig. 3.21, covering the area this MC value represents. A normal distribution fit is added (red bars).

distribution-like histograms with fitted deviations of 0.06 (second) and 0.05 (third). These results again strongly indicate the edge of chaos to have a continuous surface and a relatively plane crest.

To get a feeling for MC fluctuations within sample patches over the whole parameter space of interest, a cross-section was made acquiring MC data for the parameter sets indicated red in figure 3.25(a). For all uneven values of $k < 16$ the whole σ^2-spectrum was scanned with maximum resolution. A set F_i of 51 MC values was referred to each sample patch $A_{\mathrm{sam,i}}$. In order to combine many histograms with different average MC values the mean of each set F_i was subtracted from every element of F_i, i.e. all translated values $\tilde{mc}^i_j$ spread around zero. After this translation the memory capacities acquired in different sample patches $A_{\mathrm{sam,k}}$ were binned in one histogram. The latter can be seen in fig. 3.25(b).

Although the parts of the parameter space where the MC surface was expected to be steep were included into the cross-section histogram, i.e. both sides of the edge itself, nearly no MC value differed more than 0.2 from the mean over its sample patch. Most of them were dislocated even less than 0.1.

Hence the density of sample points across the parameter space as it was selected first can be claimed to be sensible. The MC error Δ^{sam} caused by fluctuations within one sample patch is in the same order as the error Δ^{gen} caused by many different network generations. Thus, missing significant single peaks of memory capacity within the parameter space by the sparse scanning presented above is not to be expected. As not all possible values of k

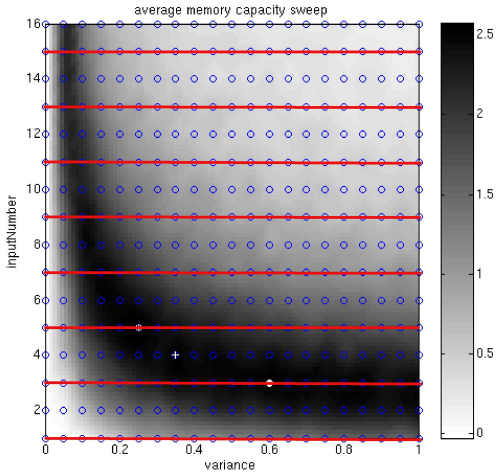
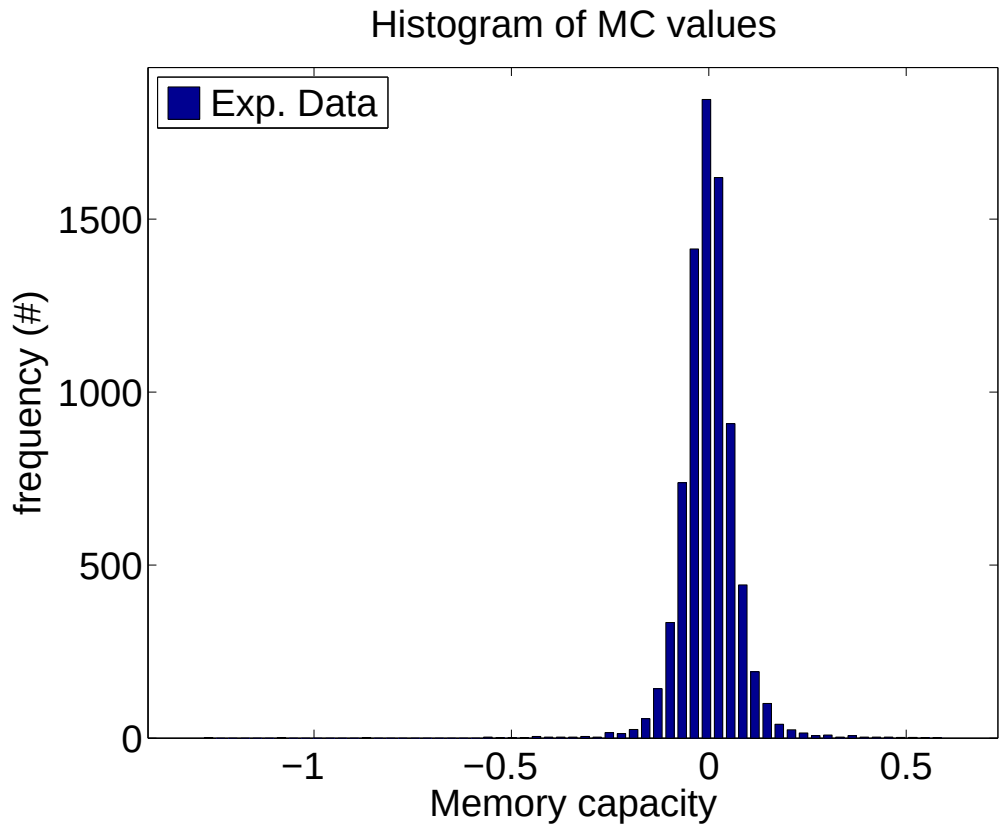


Figure 3.25: (a) Again the left plot of fig. 3.21, marking all lines of parameters for uneven values of k (vertical axis, "inputNumber"). Along these red lines MC measurements with maximum resolution were performed in order to obtain information about fluctuations of the MC surface not captured with the coarse sample grid (blue dots). The blue dots demarcate subsets of the parameter pairs defined by the red lines. Each MC value measured in such a subset was subtracted by the mean over the whole subset. A histogram of all MC values along the red lines and after application of this translation is shown in (b). The histogram proves the MC surface to be smooth in the sense of fluctuations larger than 0.2 being very unlikely. The runaway values on the left side of the peak are MC values evaluated for $\sigma^2 = 0.0$, which always have to be zero since the generated networks have no connectivity.



were evaluated ($k = 17...120$ was skipped), one might possibly miss better values of memory capacity than the ones found in the selected region. But for all Bertschinger-like experiments performed for this study the largest value for k holding a global maximum memory capacity ever measured was $k_{\max} = 12$; this was even the only case in which k was larger than 10. Hence the global MC maximum over the parameter space available with the HASTE platform always was assumed to lie within the reduced parameter region $k \leq 16$.

The histograms of MC values close to local maxima show that right on the edge of chaos Δ^{gen} dominates Δ^{sam}, so giving only Δ^{gen} as an error estimation for $\overline{\mathrm{MC}}$ is sufficient. Reducing Δ^{gen} by a factor n would cost up to $O(n^2)$ more MC evaluations in both cases, as it goes with $\sqrt{N}$ (N being the number of evaluated networks per parameter set).

In conclusion, the largest memory capacity available with P-HASTE and 120 neurons can be stated to be $\overline{\mathrm{MC}} = 2.6$ with an estimated error of $\Delta^{tot} \approx 0.05$. Claiming this is only possible if the parameters for a global maximum are assumed to lie within the sampled regions, which was strongly motivated, but not absolutely proven.

Due to the fact that the error Δ^{tot} is dominated by the statistical error Δ^{gen}, a HASTE setup has to have a $\overline{\mathrm{MC}}$ of about $0.1 = 2\Delta^{tot}$ larger than the one it is compared with to know with a confidence of 95% that it is really a better substrate for liquids.

3.2.2 MC Optimization by Input Shaping

The idea of increasing the memory capacity of an ANN liquid by temporally stretching and shaping synapses' outputs was already introduced and motivated in the beginning of this section. In what follows the way of configuring a network's synapses will be called *synaptic setup*. Assessing and comparing different synaptic setups was done using the method developed and proved for P-HASTE, i.e. finding an estimation for the maximum memory capacity $\overline{\mathrm{MC}}$ over a reasonable parameter space. This $\overline{\mathrm{MC}}$ value of a synaptic setup was then interpreted as a measure for its capability to be a substrate for liquids. Other measures are difficult to find as the position of the edge of chaos in parameter space is not fixed. Thus observing a single fixed point within parameter space and comparing MC values for different synaptic setups belonging to this fixed point makes no sense.

As a first approach the Bertschinger topology and the parameter sweep already used for P-HASTE was retained, only the answer of synapses to incoming signals and the frequency of the input stream were manipulated. The synaptic setup of HASTE has many degrees of freedom, the ones selected to change were a global transmission delay larger than zero and stretching/shaping synaptic answers. The synapses were configured to apply an exponential decay to the postsynaptic neuron if a presynaptic signal had occurred. The refractory mechanism was kept deactivated, as it significantly worsened the results for the input driven networks in all cases. A problem of stretching the synapse answers without shrinking shape amplitude or weights is the increase of total activity caused by more effective input. This leads to an earlier onset of chaotic behavior compared to applying pseudo delta peaks. Increasing synaptic time-scales relatively to the input frequency can make the network too slow to properly process the provided information. The problem was solved by temporally "blowing up" the input stream by a factor n, i.e. applying every input bit n times instead of only once (see sec. 2.1). This complies with a relative acceleration of the system by the factor n compared to the input frequency.

Systematical search

The same experiment as the one described above for P-HASTE with 120 neurons was repeated with different configurations concerning input blowup, feedback delay and time-scale of synaptic exponential decay. The amplitude resolution for synapses always was set to two bits. The input blowup factor b was set to one, two and three, for every value of b the feedback delay d_{FB} was swept from 0 ms to b ms in steps of $\Delta^{d_{\mathrm{FB}}} = 1\,\mathrm{ms}$. For every value of d_{FB}, τ was increased in steps of $\Delta^{tau} = 0.5\,\mathrm{ms}$, starting from 0.5 ms. The temporal resolution always was configured as $\rho_{\mathrm{temp}} = 1000\,\mathrm{steps/sec}$, but since there was no time-scale inherent in the experiment that would give any biological reference, this setting was arbitrary. Due to the lack of time references it makes no sense to talk about *simulated real time* at this point. Simply specifying numbers of cycles would do. But since the dimension of τ in the software GUI of HASTE is milliseconds and similar experiments to follow will be modeled on biology this unit of measurement will be kept. The ratio $\rho_{\mathrm{temp}}/\Delta^{tau}$ determines the effect of increasing τ.

All results are summarized in table 3.1. Every single number shown there represents the maximum memory capacity $\overline{\mathrm{MC}}$ across the same parameter space as for the P-HASTE experiment. The same grid of sample points has been used, 30 measurements have been performed for every evaluated point, a new network has been created randomly for every

measurement, a linear readout has been trained for every new network (2000 network cycles) and its performance has been evaluated (2000 more network cycles). Every number is the result of $\sim 40 \cdot 10^6$ network cycles and $\sim 10^4$ new networks and readout training runs. Table 3.2 gives error values to some selected $\overline{\text{MC}}$ values. These errors were obtained with the method described above, i.e. by binning the values of a highly resolved memory capacity scan across the sample patch of $\overline{\text{MC}}$ and making a normal distribution fit. The σ of this fit was interpreted as the error of $\overline{\text{MC}}$ and is shown in the table.

τ (ms) \ Blow / FB(ms)	1/0	1/1	2/0	2/1	2/2	3/0	3/1	3/2	3/3
0.5	**2.57**	0.00	2.00	2.54	0.00	1.54	2.30	2.44	0.17
1.0	2.55	0.02	2.11	**2.60**	0.18	1.65	2.46	**2.47**	0.28
1.5	1.91	0.03	2.28	2.22	0.15	2.05	2.47	2.29	0.19
2.0	1.65	0.02	**2.37**	2.10	0.14	2.08	**2.54**	2.16	0.28
2.5			2.20	2.02	0.13	2.09	2.48	2.13	0.27
3.0			1.71	1.67	0.10	**2.26**	2.31	1.98	0.22
3.5						2.22	2.08	1.95	0.20

Table 3.1: $\overline{\text{MC}}$ values for experiments according to Bertschinger, with different synaptic setups (see text for details). "Blow" denotes the temporal stretching factor of the input bit stream. "FB" stands for *feedback delay*. τ is the time constant of synapse signals' exponential decay. The maximum values in each column are marked bold (except for the columns with marginal MC values only).

τ (ms) \ Blow / FB(ms)	1/0	1/1	2/0	2/1	2/2	3/0	3/1	3/2	3/3
0.5	0.05			0.05				0.04	
1.0	0.06			0.05				0.05	
1.5			0.06	0.08			0.05	0.09	
2.0			0.07				0.04		
2.5			0.06			0.07	0.05		
3.0						0.07			
3.5						0.06			

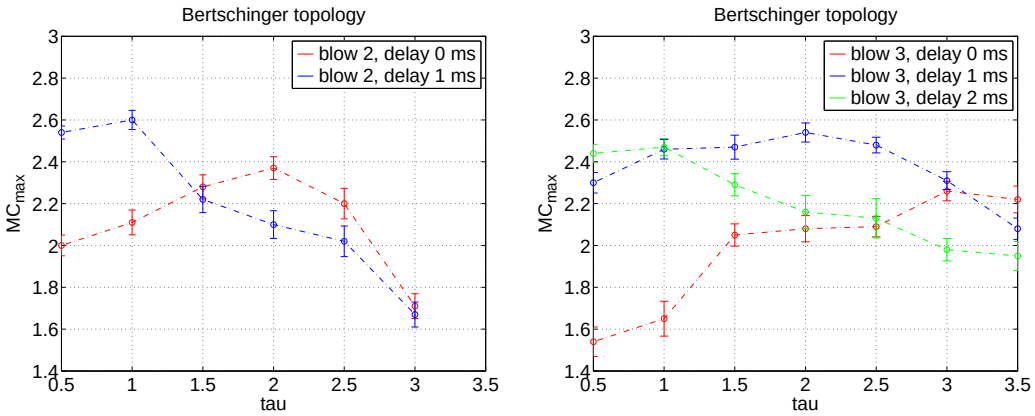
Table 3.2: Errors for the values in table 3.1, obtained with the sample patch method.

Obviously there is an optimal time constant for nearly every combination of blowup factor and feedback delay, and apparently there is a system behind these optima. Let t_{bit} be the period one single bit of input information is applied to the network during an experiment. For example in the case of a blowup factor $b = 3$ and a temporal resolution $\rho_{\text{temp}} = 1000\,\text{steps/sec}$, t_{bit} would be 3 cycles times 1 ms, i.e. $t_{\text{bit}} = 3\,\text{ms}$. The best values for $\overline{\text{MC}}$ seem to occur if

$$\tau + d_{\text{FB}} \overset{!}{\approx} t_{\text{bit}} \quad , \tag{3.24}$$

i.e. if the time of a synaptic course plus its transmission from one neuron to the next is about the period an input bit is applied.

Fig. 3.26 plots parts of table 3.1, namely the maximum memory capacity $\overline{\mathrm{MC}}$ versus the time constant τ of the exponential decay for input blowup factors two and three (ignoring experiments with $\overline{\mathrm{MC}} < 1$). The existence of an optimum depending on the input frequency b^{-1} and the feedback delay can be seen as peaks in the course of $\overline{\mathrm{MC}}$.



(a) Blowup factor 2

(b) Blowup factor 3

Figure 3.26: $\overline{\mathrm{MC}}$ versus time constant τ of exponentially decaying synaptic answers. Experiments with a time blowup of factor two are shown in (a), those with factor three in (b). In every figure the colored subplots represent measurements applying different settings of the global feedback delay (see legend).

The clear and consistent course of $\overline{\mathrm{MC}}$ as a function of τ with step sizes in the order of 0.1 supports the estimation presented above of Δ^{tot} being smaller than 0.1.

A first encouraging finding of the experiment series shown in table 3.1 is the fact that shaping synapse answers can work without worsening the capability of the system to be a liquid. This can be detected when looking at the marked optima in the rows belonging to τ values larger 0.5 ms, i.e. belonging to experimental setups with drawn-out synaptic shapes. Compare the $\overline{\mathrm{MC}}$ values for $b = 2$ complying eq. 3.24 for $d_{\mathrm{FB}} = 0$ and $d_{\mathrm{FB}} = 1$. Apparently stretching the input shapes works better if the transmission time for feedback signals is larger than zero. For distinct delays larger than zero $\overline{\mathrm{MC}}$ becomes significantly larger than in the case of no delay.

The total performance could not be optimized significantly compared to the P-HASTE experiment, but it was not worsened either by all approaches. It was possible to add more sophisticated multi-bit input shaping without losing liquid performance in the scope of computing boolean functions.

3.2.3 A Cortical Microcircuit for HASTE

In chapter 1 the consequences of High Conductance States on the capability of a Perceptron to simulate cortical neurons elaborately have been discussed. The experiments presented so far have been important to confirm basic assumptions, to develop methods for coping with the limitation of hardware resources and to gain first experiences with liquid computing. A relation to biology already was given for the single neuron bombardment experiments, but the liquid computing approach according to Bertschinger had only little to do with biological networks of spiking neurons. This is in contrast to the experiments presented in [15]. Utilizing a software simulation, the authors implemented a network of I&F neurons connected accordingly to topologies found in the rat cortex. This net then was used to act as the high-dimensional filter of a LSM. The corresponding linear readout was successfully trained to solve tasks like speech recognition.

The following experiments are the biologically inspired trial to perform spike-based real-time computing on a VLSI Perceptron. The capability of the HASTE system to perform spike-based computation close to cortical activity was used to full capacity. Network topologies were created strongly oriented towards real cortical structures, according to the instructions given in [15]. The mapping of these instructions to the parameter space provided by HASTE was introduced in sec. 2.2.1. The neuron and synapse parameters also were chosen to fit data suggested in [15].

Typical stimuli in real life are highly continuous. Hence, the three-bit parity problem on a binary input stream in discrete time domain used so far is not an optimal benchmark problem for the attempt to perform biological computation. But since the whole experimental setup already existed and only small adaption had to be performed to apply this problem to the new HASTE setup, it was worth a try. Furthermore, the human brain indeed can solve such a task.

Applying a new network topology

A network of 120 neurons formed the liquid of a LSM, the synapses were simulated with $\rho_{\text{amp}} = 2$. The neurons were arranged in a virtual three-dimensional integer grid with size $6 \times 5 \times 4$ and then connected according to eq. 2.6, see sec. 2.2.1 for details. The parameters $\tilde{A}$ determining the relative connectivity between different neuron types were selected as suggested in [15], namely $C_{EE} = 0.3$, $C_{EI} = 0.2$, $C_{IE} = 0.4$ and $C_{II} = 0.1$. The suggestions for relative synaptic strengths in these different cases had to be adapted to the requirements of HASTE. An elaborate derivation can be found in sec. 2.2.1, too. The adapted values chosen for the experiments are $\tilde{A}_{EE} = 0.499$, $\tilde{A}_{EI} = 1.000$, $\tilde{A}_{IE} = -0.887$ and $\tilde{A}_{II} = -0.887$. The value for λ, which determines the total connectivity, was swept during the experiments to follow. The response of synapses to arriving spikes was configured according to eq. 3.10, i.e. it was modeled as an exponential decay. 20% of the neurons were set inhibitory, 80% excitatory. The whole setup worked with a temporal resolution of $\rho_{\text{temp}} = 1000\,\text{steps/sec}$.

The time constant of the exponential decay of synaptic response was set to 6 ms for inhibitory and to 3 ms for excitatory synapses. The discretization method CCI proposed in sec. 3.1.1 was applied. Since only a global (i.e. for all neurons identical) refractory period can be defined within the hardware version of HASTE, it was set to 2 ms. The global transmission delay of feedback connections was set to 1 ms.

Injecting the input bit streams

The linear readout of the LSM was trained to solve the three bit parity problem like already introduced for the Bertschinger-like experiments. Again both the original bit stream, its inverse and one constant bias input were fed into the network. The speed of information inflow was adapted to the synaptic time constants by blowing up the primarily generated input bit stream with zeros (see sec. 2.2.2). Three zeros followed every bit that contained information, i.e. every 4 ms of simulated time a new bit was applied to the network. The bias input was received by every single neuron. But in contrast to the input-driven network suggested by Bertschinger, for this cortical microcircuit the two input bit streams were not stringently connected to every neuron from the start. Eq. 2.6 describes probabilities for connections between inner neurons, i.e. both participating neurons are part of the simulated network. The equation can be defined separately for connections from external inputs to the network itself. Then each of the appearing parameters is marked with a subscript *in*. Corresponding to the network activation applied in the Bertschinger-like experiments, the synaptic strength was set to 0.5 hwu for injecting the original bit stream, and to -0.5 hwu for the inverse signals. C_{in} was set to 1.0 for all connection types, i.e. all neurons had the same chance to receive an input. λ_{in} determined the total degree of input connectivity. For values of $\lambda_{\text{in}} > 10$ practically every neuron receives both bit streams. Therefore sweeping λ_{in} does not have to exceed a value of 10.

Results

Since the network topology suggested by Maass provides a very large number of parameters, a systematical search like the one proposed for the Bertschinger-like networks was not possible due to the limited time for the work on this thesis. Two experiments are presented here, they are to be understood as a proof of principle.

Again two-dimensional parameter sweeps were performed. Both the average memory capacity and the course of the hamming distance have been evaluated in the same way as for the previous experiments. The number of evaluations per data point again was 30. For the first run λ_{in} was set to infinity, i.e. every neuron received both input bit streams. The synaptic weight of the bias input was the first parameter to vary, the complete available range was covered. This value shifts the effective simulated spiking threshold of all neurons and therefore directly affects the network activity. The second parameter to be swept was λ, determining the total connectivity of the net and consequently an important influence on the activity, too. The evaluated values range from 0.25 to 5.0.

Fig. 3.27 shows the result. Obviously a biasing of the neurons with too large negative values causes a complete loss of memory capacity. Since one input streams is injected with synaptic weights of 0.5 hwu, the onset of memory capacity for bias weights larger than -0.5 hwu was expected. Another edge of systematic MC changes can be detected for bias weights of 0.5 hwu. This corresponds to the synaptic weights the second bit stream is fed in with, namely -0.5 hwu.

Much more interesting is the clear and finite range of large memory capacity determined by λ. Values of 1.3 ± 0.3 are found to work best, at least for a negative bias weight larger than -0.5.

In accordance with the results for the Bertschinger-like experiments, the figure again indicates that the memory capacity of a network correlates to the onset of chaotic activity.

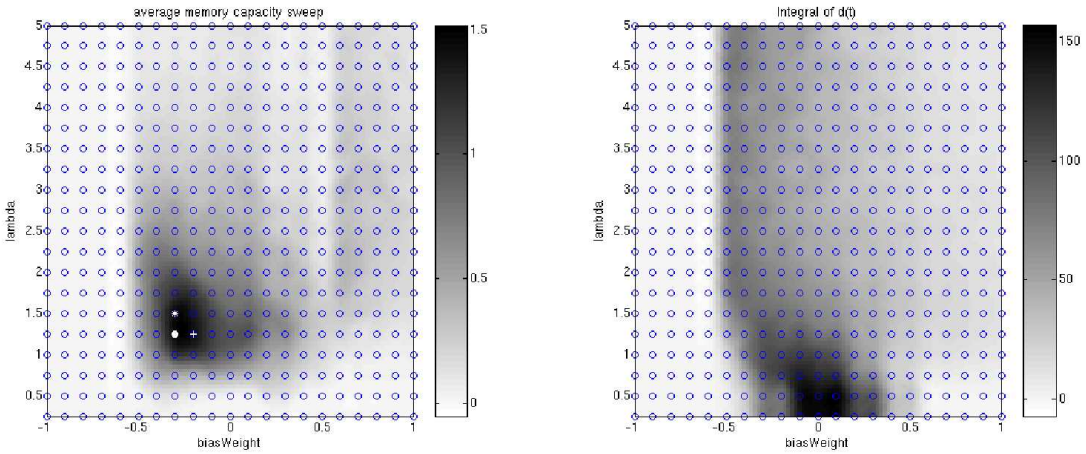


Figure 3.27: Left: Average memory capacity (grey value) of liquids generated according to Maass as a function of two parameters, namely the synaptic weight of the bias input (axis "biasWeight") and λ. Right: Integral over the hamming distance of two identical networks fed with the same input bit stream after a phase of different initialization, again as a function of the same two parameters. This integral is considered to be a measure for the degree of chaos within the network activity, see sec. 3.2.1. Clearly for bias weights less than -0.5 the network exhibits highly ordered respectively probably even no activity. For bias weights right next to this critical value chaos is strongest. It disappears again for increasing values of this weight. This probably comes along with the permanent firing of more and more neurons due to the large biasing. A finite region is dominated by high integral values, in particular for bias weights around zero. Obviously there is a correlation again between the memory capacity and the degree of chaos.

One consequence of the first experiment with the cortical microcircuit was, that for the experiment to follow a bias weight of -0.4 hwu has been selected. For the same basic settings a parameter sweep λ vs. λ_{in} was performed, λ_{in} being varied from 1 to 10. The goal was to find out whether the new topology possibly works better if less neurons receive the external input. Possibly the strong input driving inhibits some neurons from performing tasks that require a high input sensitivity. Both the average memory capacity and the hamming integral has been evaluated for every data point.

Fig. 3.27 shows the result. The small range of values for λ causing a high memory capacity is confirmed, it can be seen as a band of high average MC values going from the left to the right. Again this band corresponds to the onset of a higher degree of chaotic activity. The first maximum of the memory capacity is found to be 1.49 ± 0.04, which is significantly smaller than the value found for the Bertschinger approach with 120 neurons. Nonetheless this value denotes the actual capability of the liquid to store information and to act as a non-linear filter.

Since the memory capacity within the dark band seems to be nearly constant for a large spectrum of λ_{in}, a cross-section is plotted in fig. 3.29. Obviously the network is very tolerant in its response to different degrees of input drive. The average memory capacity already reaches its maximum value for a fragmentary connection of the input bit streams to the network.

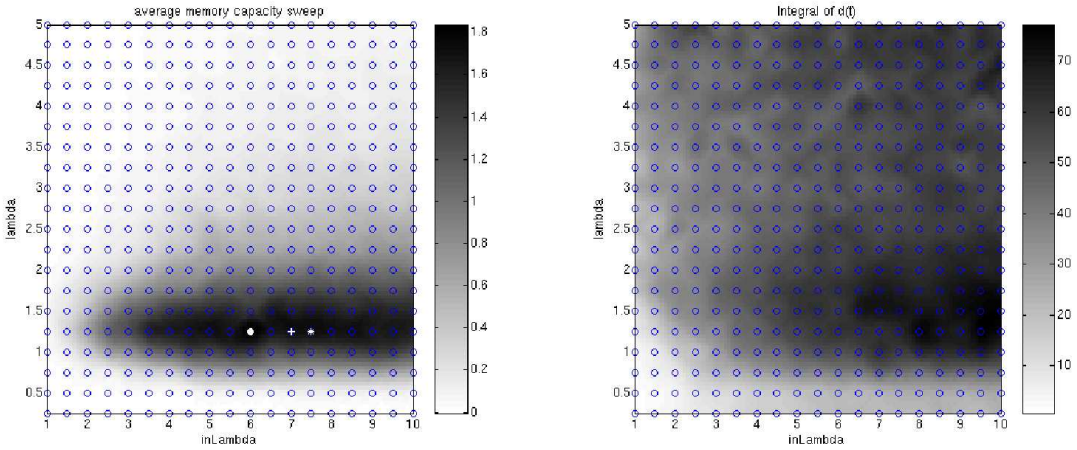


Figure 3.28: Left: Average memory capacity (grey value) of liquids generated according to Maass as a function of two parameters, namely the synaptic weight of the bias input (axis "biasWeight") and λ. Right: Integral over the hamming distance,

From this point the average MC keeps approximately constant when increasing the input connectivity, at least within the occurring fluctuations. The maximum value found for the memory capacity in this experiment is 1.84 ± 0.04, which is significantly higher than for the λ vs. bias weight sweep.

Conclusions drawn from this section

HASTE has been utilized for liquid computing in different ways. First a typical Perceptron experiment has successfully been reproduced, proving HASTE to be able to solve real-time tasks. The correlation between chaotic activity of a network and its memory capacity elaborately has been studied. Then features provided by HASTE to extend the Perceptron model have systematically been applied. A way to reproducibly optimize the maximum memory capacity a HASTE configuration can exhibit was not found. Nonetheless the memory capacity could be upheld even for synaptic shaping and stretching.

A cortical microcircuit has been constructed on the basis of the HASTE system. The same problem as for the first approach was to be solved with really spike-based computation. For some regions in configuration space the neural network exhibited a significant reproducible memory capacity in the same order as for the pure Perceptron approach. Therefore the attempt to perform spike-based real-time computation on the basis of a hardware Perceptron can be claimed to be successful.

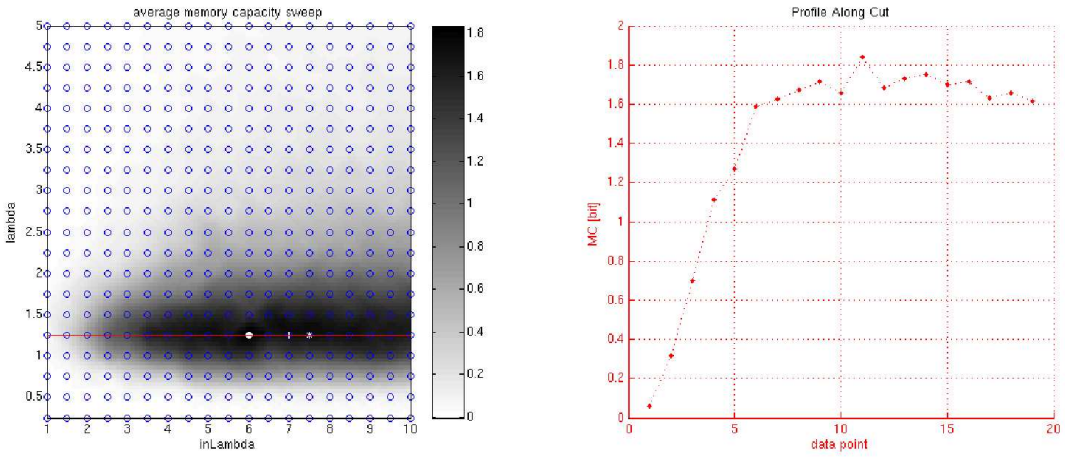


Figure 3.29: Left: Same as left part of fig. 3.28, but including a red line that marks all parameter points the memory capacity is separately plotted in the right figure. Right: Average memory capacities of the points cut by the red line in the left figure.

Chapter 4

Discussion

A way has been found to utilize a Perceptron for the simulation of cortical neural networks in awake state. This approach is based on the interpretation of the neurons' signals as being spike-coded and can be motivated consulting recent results from neuro-science, especially those of so-called High Conductance States of neuron populations. The simulation method was applied to the VLSI Perceptron HAGEN, which consists of 256 McCulloch-Pitts neurons. The design implemented for this study is therefore called HASTE (HAGEN spike translation environment).

To extend the hardware neurons with basic features of a conductance-based spiking neuron model, functional units have been designed for the programmable logic device (FPGA) which controls the HAGEN chip. The implementation was realized by the tutor of this study, Michael Reuss, who also took part in the planning. An essential part of this extension is the pooling of multiple binary Perceptron synapses to one multi-bit synapse with a limited range of integer values. This allows to model shaped synaptic response to arriving spikes with a certain resolution. The devices specifically designed for HASTE and realized on the FPGA perform the generation and application of these shaped inputs as well as the realization of neuron refractoriness. The generators are constructed to model the change of simulated membrane conductances at synapses due to arriving spikes. The McCulloch-Pitts neurons superpose these conductance courses and discriminate whether the ratio of excitatory to inhibitory components exceeds a critical value. If this happens, the neuron exhibits an output value of 1 and is said to fire at a certain rate until the output vanishes again.

The HASTE system was integrated into a software framework that already existed for the controlling and utilization of the HAGEN chip. Therefor classes representing the new network model, the shape generators and others have been written. Methods for generation of biologically oriented network topologies and their mapping to the available hardware have been developed. Many algorithms and methods had to be adapted to the special requirements of synapse pooling in order to apply them to experiments performed for this study.

HASTE is working, the software is well documented and the hardware extensions are described and evaluated in this thesis. A software version of the whole system has been developed by the author in the design-phase of the hardware realization, referred to as SoftHASTE. SoftHASTE already utilizes the real HAGEN chip as its Perceptron core. It allowed to explore and cut down the huge parameter space that arose from first conceptual planning due to its huge flexibility. In contrast to re-programming the FPGA, no constraints in terms of limited hardware resources have to be considered for this software extension. It therefore was used

to support Michael Reuss in his work on the hardware implementation.

Three different types of experiments have been performed with the novel HAGEN environment. In the first run single neuron bombardments were simulated using SoftHASTE. Methods have been found to discretize conductance courses allowing to compare experimental results obtained with different amplitude and temporal resolutions in terms of output firing rates. Emulating continuous membrane dynamics of a neuron with 1000 discrete cycles per simulated second was shown to deliver reasonable output firing rates. These first experiments proved the ability of HASTE to process data via *rate coding*.

A second point to test was the distribution of interspike intervals exhibited by a HASTE neuron under biologically realistic input. A significant ISI diversity is essential for *temporal coding*, a method of information processing that is found within the brain. HASTE implements only an absolute refractory mechanism for the time being. In spite of this constraint the results of the second sort of experiments demonstrate that it is possible to configure HASTE such that the interspike intervals spread in the same order as data retrieved during in vivo measurements of real neurons, at least for a realistic input bombardment.

In a third type of experiments neuron populations were simulated. A liquid state machine developed by a member of the author's working group was adapted to HASTE, integrating the spike-based neural network as the high-dimensional filter component. The task to solve by this system was a three-bit parity problem on a continuous one-dimensional bit stream fed into the network. The three bits the parity had to be solved for were systematically shifted backwards in time, i.e. the capability of the network to keep information (memory capacity) was tested. In a first attempt, a setup emulating a plain Perceptron and a procedure proposed by Nils Bertschinger et al. was applied to HASTE respectively to the liquid state machine. It was shown that the HASTE environment configured to behave like a plain Perceptron reproduced previous results of liquid experiments with HAGEN only performed by Felix Schürmann [28].

Different attempts have been made in order to optimize the memory capacity of the Bertschinger-like network. Shaping and stretching synaptic answers to arriving spikes was varied in a systematic search. The ability of the HASTE network to deliver equal memory performance with elaborate synaptic response as for the pure Perceptron approach has been proven. A significant improvement in terms of memory capacity was not found.

In another approach to solve this real-time computation task the capabilities of the HASTE system were used to full capacity for the first time. The network and each single neuron were configured to act as close to real cortical structures as possible. Main reference for the selection of parameter values was a publication by Wolfgang Maass about liquid computing experiments with a software model of cortical spiking neurons. The benchmark problem was not ideal for testing a cortical circuit, since it was on a one-dimensional input, which is in contrast to the typically multi-dimensional appearance of real stimuli in the brain. The digital character of the task does not really fit the pseudo-continuous operation mode of the network either. But in spite of this drawback, the cortical microcircuit for HASTE exhibited a memory capacity in the same order as the Perceptron approach with a Bertschinger-like network topology. Hence, spike-based real-time computation has been performed on a hardware Perceptron with biologically oriented extensions.

In conclusion, the design and implementation of a spike interpretation environment for a hardware Perceptron was successful. Powerful and flexible software has been created to comfortably utilize this novel tool. The experiments performed with SoftHASTE and HASTE delivered results of different usefulness. A basic and important insight is that HASTE is able to perform both rate coding and temporal coding. Spike-based real-time computation with

generic cortical circuits was proven to be possible on this platform. Thus, a good starting position for further experiments exceeding the field of system specification has been created.

Chapter 5

Outlook

Generally, the main advantage of using hardware for the simulation of neural networks clearly is the computational speed and the high scalability due to operation parallelism. Hardware approaches that implement neuron models exhibiting spiking behavior are very rare. In contrast to that, several hardware Perceptron devices exist. Utilizing these devices for the simulation of networks of spiking neurons provides a fast tool for studying basic mechanisms of the brain like information representation and self-organization. In spite of the limited accuracy, the enormous simulation speed and thus the possibility of simulating large periods of activity might make this approach the first choice in some cases.

A long term application of HASTE is the research on spike-timing dependent plasticity of synaptic weights. Reasonably modeling and studying this way of synaptic learning promises to give new insights into basic mechanisms of self-organization within the brain.

A very concrete plan of progressing in the field of liquid computing is to perform experiments with more naturally inspired tasks to be solved by the liquid state machine. Candidates are simple speech recognition problems or the classification of sequences of visual stimuli.

Before approaching these goals, some technical improvements should be considered. Although the setup presented in this study marks a certain status of completion, there is a number of restrictions which possibly can be eliminated or reduced. The refractoriness of neurons for example can only be configured with a global value for the whole network. This is not in accordance with the diversity of refractory periods found in nature. At least two different periods should be realizable since a main distinction in real cortical networks is the difference of values found for excitatory and inhibitory neurons. A relative refractory mechanism is planned, but not yet realized. For the time being many neurons on the chip have to be ignored if the synapse resolution is more than one bit. These neurons can be used to deliver information about the closeness of the simulated conductance ratio to the critical value. This information might be the basis for a relative refractoriness.

The transmission delay of feedback connections can only be selected globally as well. It is furthermore limited to a maximum value of three network cycles, which makes this limit even inconsistent in terms of time for different temporal resolutions. Both constraints are due to hardware specific reasons, but they might be solvable with a larger FPGA, for example.

Maybe the most urgent improvement that has to be taken care of is creating the possibility of a network reset. This means the need for setting all conductance course generators and all information stored in the feedback registers to zero. Since this feature is missing for the time being, time-consuming initialization phases had to be applied before the evaluation of

network states was possible.

The HAGEN chip was designed as a prototype for a larger network device. This bigger version with a synapse number in the order of 10^6 is still planned to be produced, especially since HAGEN proved to work very reliably. The developments made for HASTE easily could be expanded and adapted to such a larger chip, partly solving the problem of the neuron number limitations coming along with increased synapse resolution.

All things considered, much of the potential of HASTE still has to be exploited.

Appendix A

Supplement

A.1 Additional figures

Belonging to sec. 2

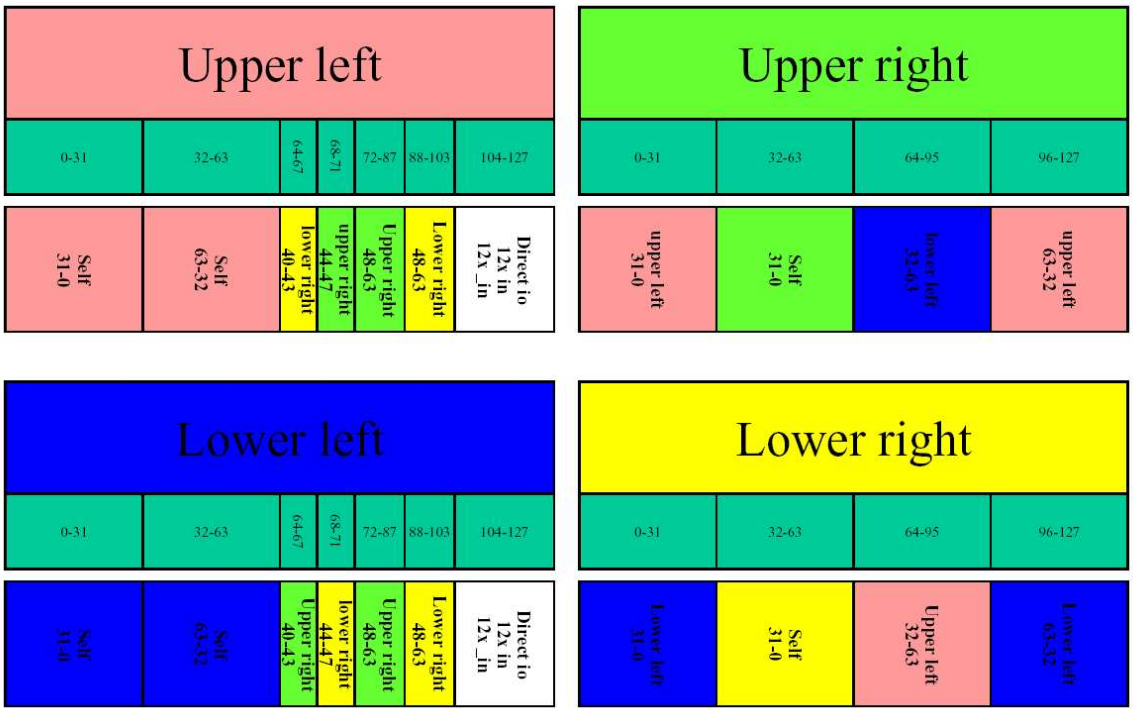
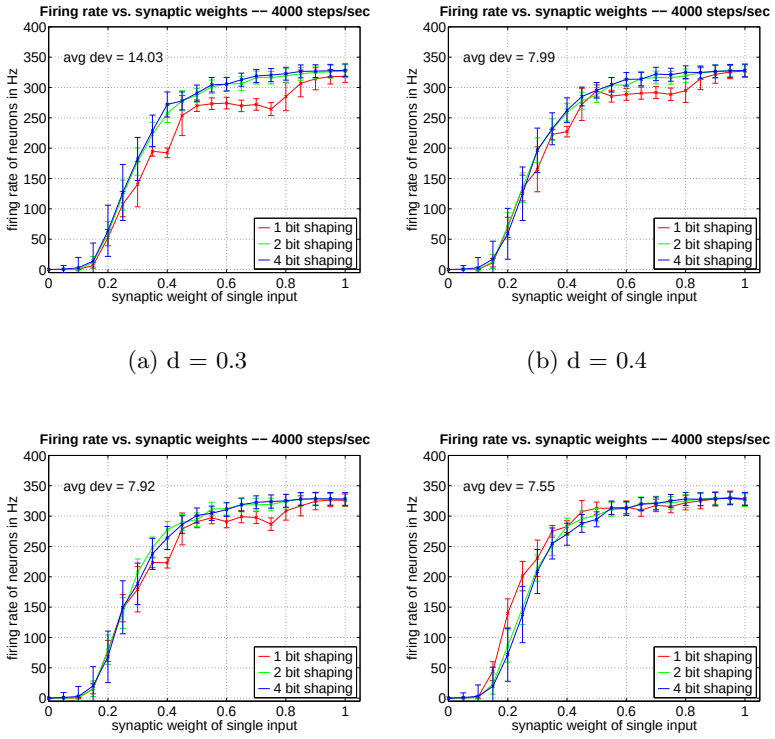


Figure A.1: HAGEN feedback connection overview.

Belonging to sec. 3.1.1



(a) d = 0.3

(b) d = 0.4

(c) d = 0.5

(d) d = 0.6

Figure A.2: Output firing rate of a single neuron bombarded with 30 Poisson spike trains versus the synaptic weight of a single input (all 30 inputs have the same weight). Each input spike train has an average firing rate of 33 Hz, the temporal resolution is 4000 steps/sec. (a)-(d) Mapping offset d constant for all bit resolutions, (c) with $d = 0.5$ represents arithmetic rounding.

Belonging to sec. 3.2.1

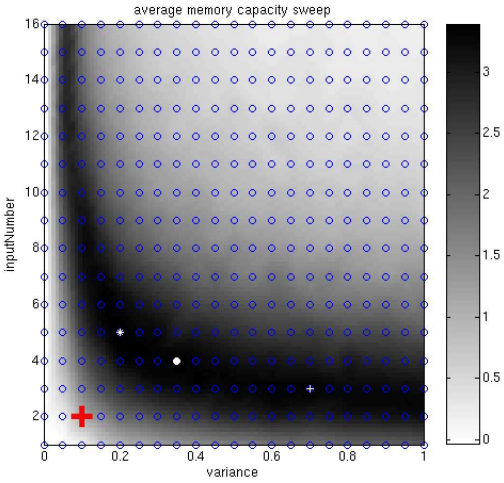
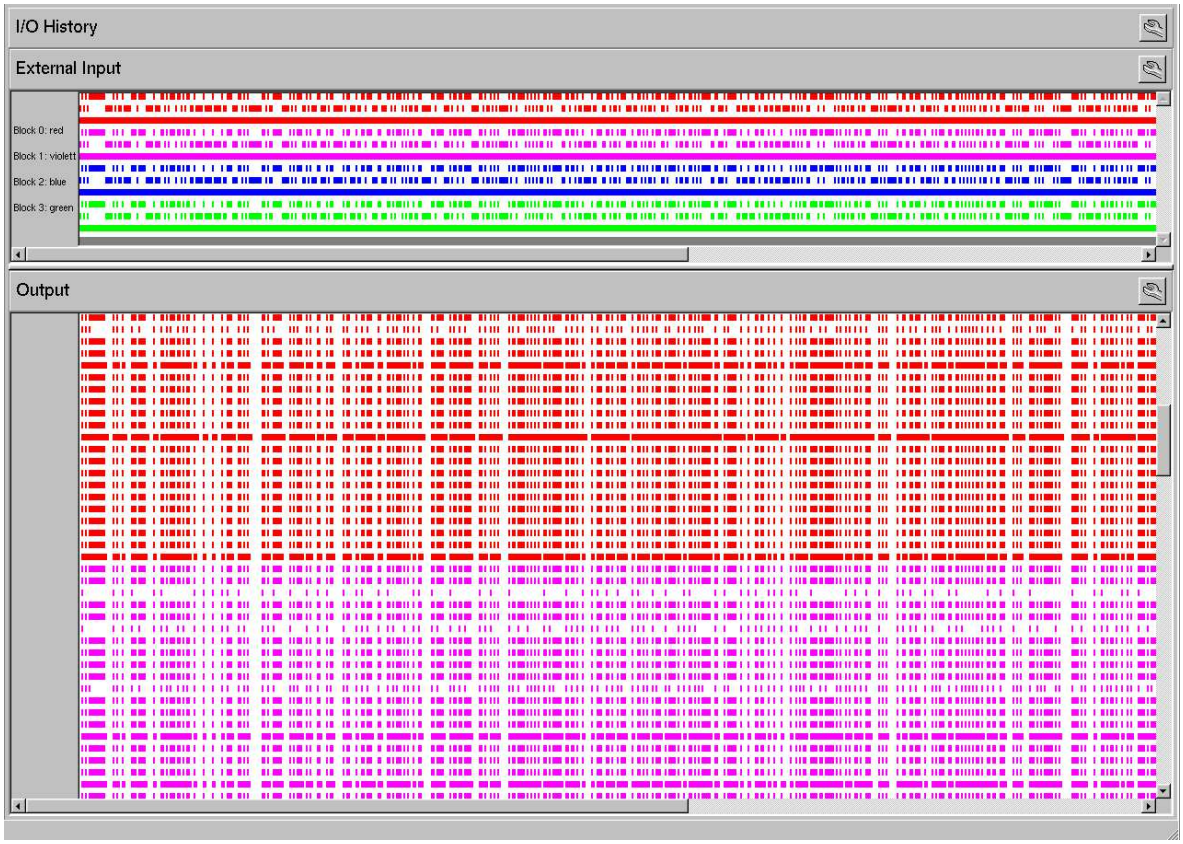


Figure A.3: The grey-scale coded MC values to the left represent the data retrieved during the 256 neuron P-HASTE experiment. The red plus marks a point within the parameter space below the band of high memory capacity. According to these parameters a network was generated using the same experimental setup (P-HASTE, 256 neurons). Just like for the MC evaluation every neuron was connected to a random bit stream, it's inverse and a constant bias. This can be seen in the "input" sub-window of the lower figure: each of the four network blocks (colored) receives the described input. The synapse weight connecting the bias with the neuron was set to zero. The resulting output versus time is shown in the lower sub-window, it strongly follows the input. The output window does not show all 256 neurons.



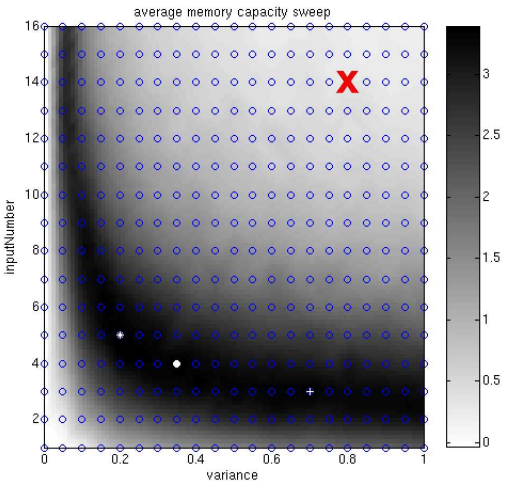
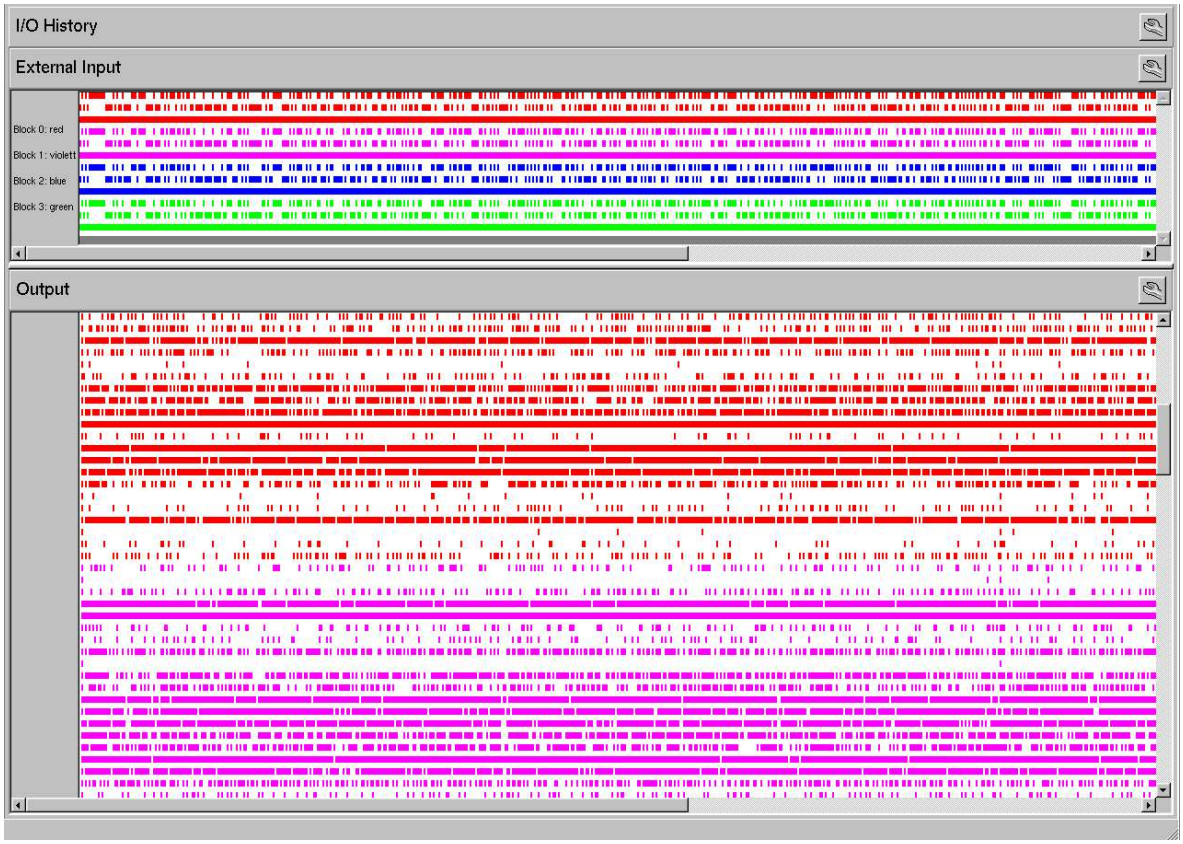


Figure A.4: The grey-scale coded MC values to the left represent the data retrieved during the 256 neuron P-HASTE experiment. The red cross marks a point within the parameter space above the band of high memory capacity. According to these parameters a network was generated using the same experimental setup (P-HASTE, 256 neurons). Just like for the MC evaluation every neuron was connected to a random bit stream, it's inverse and a constant bias. This can be seen in the "input" sub-window of the lower figure: each of the four network blocks (colored) receives the described input. The synapse weight connecting the bias with the neuron was set to zero. The resulting output versus time is shown in the lower sub-window. The activity seems to much more chaotic than that shown in fig. A.3. Again the output window does not show all 256 neurons.



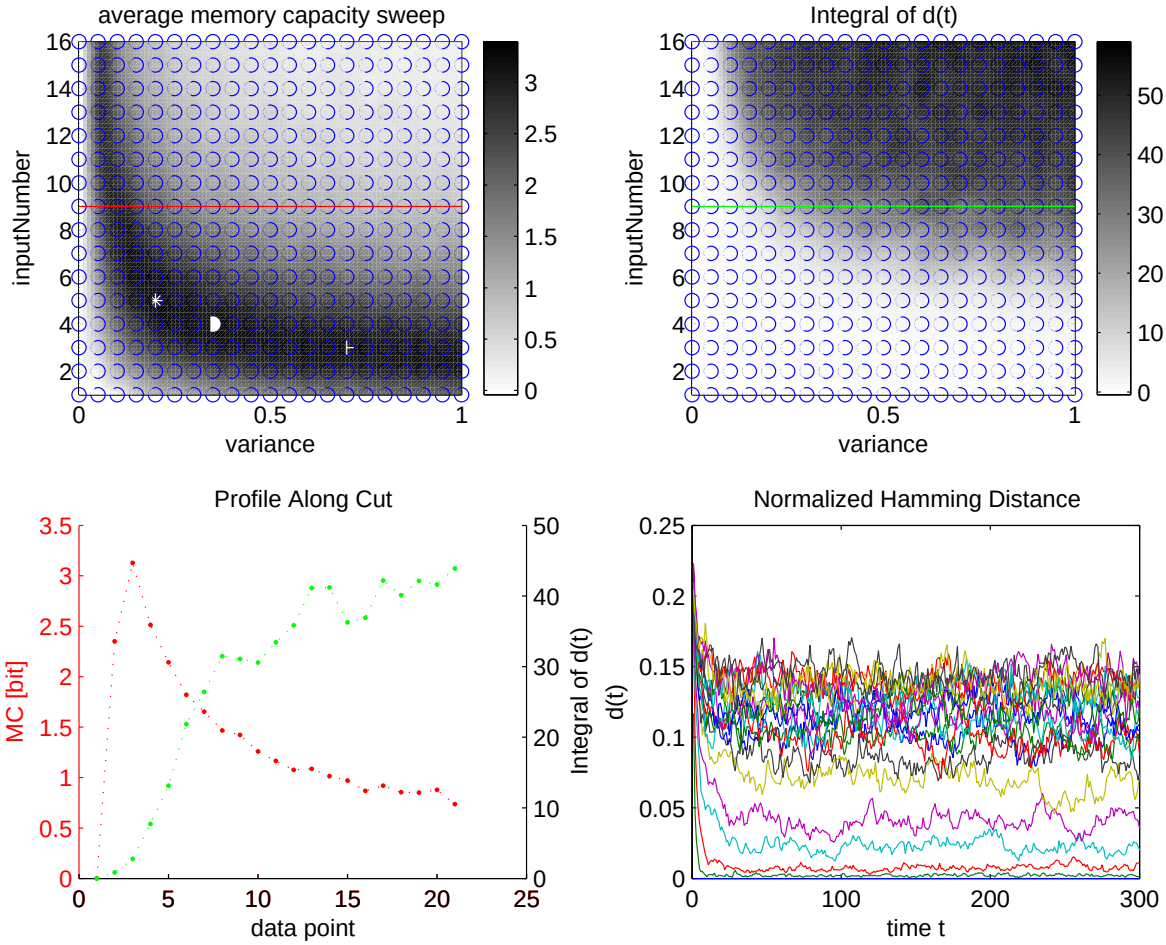


Figure A.5: Upper left: Same as left part of fig. 3.20. For the parameter pairs on the red line the memory capacity is re-plotted in the lower left figure (red). In another experiment a single network was generated for every parameter point the memory capacity was evaluated at (blue circles). This network was fed with the same input bit stream twice, being differently initialized in each case. The hamming distance $d(t)$ between the generated output states was recorded. The lower right figure shows the course of this hamming distance for every parameter point which is crossed by the green line in the upper right plot. More precisely, every plot in the lower right figure represents the mean over 30 courses evaluated for a fixed parameter set. Lower courses of $d(t)$ correspond to lower values of k. Every data point in the upper right plot denotes the integral over such a mean course. The integrals of all parameter pairs crossed by the green line (same as for the red one) are re-plotted in the lower left figure, too.

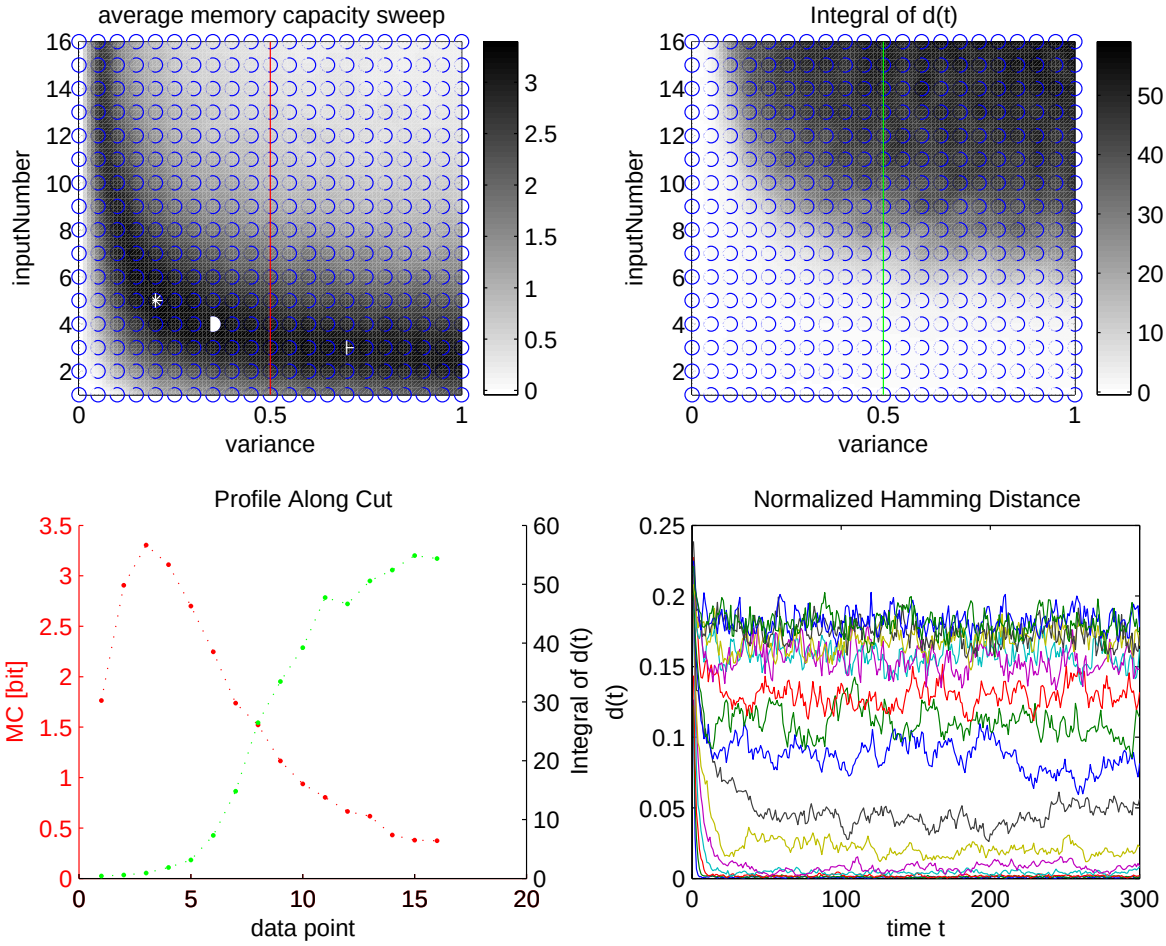


Figure A.6: Upper left: Same as left part of fig. 3.20. For the parameter pairs on the red line the memory capacity is re-plotted in the lower left figure (red). In another experiment a single network was generated for every parameter point the memory capacity was evaluated at (blue circles). This network was fed with the same input bit stream twice, being differently initialized in each case. The hamming distance $d(t)$ between the generated output states was recorded. The lower right figure shows the course of this hamming distance for every parameter point which is crossed by the green line in the upper right plot. More precisely, every plot in the lower right figure represents the mean over 30 courses evaluated for a fixed parameter set. Lower courses of $d(t)$ correspond to lower values of k. Every data point in the upper right plot denotes the integral over such a mean course. The integrals of all parameter pairs crossed by the green line (same as for the red one) are re-plotted in the lower left figure, too.

A.2 Software

This section gives short information about the location and installation of all software described in this thesis.

A.2.1 Compiling HANNEE

In order to run the HANNEE software, one has to check out the *hannee++* tree first, which can be found in the *project* directory of the CVS (concurrent version system) repository of the *Electronic Vision(s)* group. The README file included in the *hannee++* folder gives detailed instructions how to install the HANNEE program. It also contains information about the calibration of the HAGEN chip, how to generate the documentation with *Doxygen* and how to integrate personal software components into the HANNEE framework. The latter has to be done with the modules developed for HASTE.

A.2.2 Compiling the Software for HASTE

In order to make use of the software interface and the methods provided for HASTE, a proper version of HANNEE has to be installed first, see previous section. All necessary source code for HASTE can be downloaded from the *project* directory of the CVS repository of the *Electronic Vision(s)* group, too. The subtrees *myhannee/bruderD* and *myhannee/felix* have to be checked out. The folder *bruderD* contains a file *HowToGenerateLiquidHaste*, which is a step-by-step description of what to do in order to obtain a working program.

A.3 Experimental Raw-Data and Script-files for MATLAB

All experimental data accumulated for this thesis is stored on the host computer *orion* at the Kirchhoff Institute for Physics in Heidelberg. The exact path where to find it is *orion:/users/bruederl/DATA/EXP*. All script-files for the application of MATLAB to the raw data are stored in sub-folders called *./matlabScripts*.

List of Abbreviations

ANN	artificial neural network
ARN	arithmetic rounding
ASIC	application specific integrated circuit
CC	conductance course
CCI	constant course integral
CVS	concurrent version system
DAC	digital to analog converter
EEG	electroencephalogram
fMRI	functional magnetic resonance imaging
FPGA	field-programmable gate array
GUI	graphical user interface
HAGEN	Heidelberg analog evolvable network
HANNEE	Heidelberg analog neural network evolution environment
HASTE	HAGEN spike translation environment
HCS	High Conductance State
HWU	HAGEN weight unit
I/O	in- and output
LSM	liquid state machine
LVDS	low voltage differential signal
MC	memory capacity
MI	mutual information
PC	personal computer
PCI	peripheral component interconnect
PSP	post-synaptic potential
RAM	random access memory
STDP	spike-timing-dependent plasticity
VHDL	virtual hardware describing language
VLSI	very large scale integration
VSD	voltage-sensitive dye

Bibliography

[1] N. Bertschinger and T. Natschläeger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413 – 1436, July 2004.

[2] Alain Destexhe, Michael Rudolph, and Denis Pare. The high-conductance state of neocortical neurons in vivo. *Nature Reviews Neuroscience*, 4:739–751, 2003.

[3] Wulfram Gerstner and Werner Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[4] D. Goldenholz. Liquid computig: A real effect. Technical report, Boston University Department of Biomedical Engineering, 2002.

[5] Robert Hecht-Nielsen. Perceptrons. Technical report, Institute for Neural Computation, University of California, San Diego, 2004.

[6] S. Hohmann, J. Schemmel, F. Schürmann, and K. Meier. Exploring the parameter space of a genetic algorithm for training an analog neural network. In W.B. et. al. Langdon, editor, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2002*, pages 375–382. Morgan Kaufmann Publishers, July 2002.

[7] S.G. Hohmann, J. Fieres, K. Meier, J. Schemmel, T. Schmittz, and F. Schürmann. Training fast mixed-signal neural networks for data classification. In *Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN'04)*, pages 2647–2652. IEEE Press, 2004.

[8] Steffen Hohmann. Personal communication, Oct. 2004.

[9] Steffen G. Hohmann, Johannes Schemmel, Felix Schürmann, and Karlheinz Meier. Predicting protein cellular localization sites with a hardware analog neural network. In *Proceedings of the Int. Joint Conf. on Neural Networks*, pages 381–386. IEEE Press, 2003.

[10] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.

[11] C. G. Langton. Computation at the edge of chaos. *Physica D*, 42, 1990.

[12] Volker Lindenstruth. Informatik II (Technische Informatik), lecture notes, University of Heidelberg, summer term 2004.

[13] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10:1659–1671, 1997.

[14] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[15] W. Maass, T. Natschläger, and H. Markram. On the computational power of circuits of spiking neurons. *Journal of Physiology (Paris)*, (in press), 2004.

[16] W. Maass, T.Natschlger, and H.Markram. *Computational models for generic cortical microcircuits*, chapter 18, pages 575–605. Number ISBN 1-58488-362-6. J. Feng, Boca Raton, 2004.

[17] J. Madrenas, E. Alarcon, J. Cosp, and J.M. Moreno. Vlsi design of a flexible-structure sequential mixed-signal neural processor. In *Proceedings of the 6th International Conference Mixed-Signal Design of Integrated Circuits and Systems (MIXDES'99), Krakw (Poland), June1999*, 1999.

[18] D.A. McCormick, Z. Wang, and J.R. Huguenard. Neurotransmitter control of neocortical neuronal activity and excitability. *Cerebral Cortex*, 3:387–398., 1993.

[19] J. Montalvo, Gyurcsik R., and Paulos J. An analog vlsi neural network with on-chip perturbation learning. *IEEE Journal of Solid-State Circuits*, 32(4):535–543, April 1997.

[20] Dominik Niedenzu. Aufbau eines binären Neocognitrons. Diploma Thesis (german), Heidelberg University, HD-KIP-03-11, 2003.

[21] Carl C. H. Petersen, Amiram Grinvald, , and Bert Sakmann. Spatiotemporal dynamics of sensory responses in layer 2/3 of rat barrel cortex measured in vivo by voltage-sensitive dye imaging combined with whole-cell voltage recordings and neuron reconstructions. *The Journal of Neuroscience*, 23(3):1298 –1309, February 2003.

[22] Daniel S. Reich, Ferenc Mechler, Keith P. Purpura, and Jonathan D. Victor. Interspike intervals, receptive fields, and information encoding in primary visual cortex. *The Journal of Neuroscience*, 20(5):1964–1974, March 2000.

[23] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[24] F. Rosenblatt. Perceptron simulation experiments. In *Proceedings of the IRE*, pages 301–309, 1960.

[25] S. Satyanarayana, P. Tsividis, and H.P. Graf. A reconfigurable vlsi neural network. *IEEE Journal of Solid-State Circuits*, 27(1):67–81, January 1992.

[26] J. Schemmel, S. Hohmann, K. Meier, and F. Schürmann. A mixed-mode analog neural network using current-steering synapses. *Analog Integrated Circuits and Signal Processing*, 38(2-3):233–244, 2004.

[27] T. Schmitz, S. Hohmann, K. Meier, J. Schemmel, and F. Schürmann. Speeding up Hardware Evolution: A Coprocessor for Evolutionary Algorithms. In Andy M. Tyrrell, Pauline C. Haddow, and Jim Torresen, editors, *Proceedings of the 5th International Conference on Evolvable Systems ICES 2003*, pages 274–285. Springer Verlag, 2003.

[28] F. Schürmann, K. Meier, and J. Schemmel. Edge of Chaos Computation in Mixed Mode VLSI - "A Hard Liquid". In *Conference on Neural Information Processing 2004 (NIPS04) - accepted*, to be published.

[29] Felix Schürmann, Steffen G. Hohmann, Karlheinz Meier, and Johannes Schemmel. Interfacing binary networks to multi-valued signals. In *Supplementary Proceedings of the Joint International Conference ICANN/ICONIP*, pages 430–433. IEEE Press, 2003.

[30] Michael Shelley, David McLaughlin, Robert Shapley, and Jacob Wielaard. States of high conductance in a large-scale model of the visual cortex. *J. Comp. Neurosci.*, 13:93–109, 2002.

[31] S. Song, K. Miller, and L. Abbott. Competitive hebbian learning through spiketiming-dependent synaptic plasticity. *Nat. Neurosci.*, 3:919–926, 2000.

[32] R.J. Staba, C.L. Wilson, I. Fried, and J. Enge Jr. Single neuron burst firing in the human hippocampus during sleep. *Hippocampus*, 12:724–734, 2002.

[33] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural Networks*, 14:715–725, 2001.

[34] P. White, B. Biskup, J. Elzenga, U. Homann, G. Thiel, F. Wissing, and F. Maathuis. Advanced patch-clamp techniques and single-channel analysis. *Journal of Experimental Botany*, 50:1037–1054, 1999.

Acknowledgments

I want to express my gratitude to all who made this work possible, especially:

My father for supporting me in all I ever aimed for.

Prof. Dr. Karlheinz Meier for accepting me into the research group, for his friendly direction, his interest and his support.

Michael Reuss for being a patient, helpful and humorous tutor and partner for this work.

Felix Schürmann for his care, interest and support, for generously sharing his knowledge and tools, for pushing the project.

Thorsten Maucher for Latex support and his special friendliness.

Jörg Langeheine for introducing me into the research group, for patiently teaching me basics of experimental computer science and for his helpful hints.

All members of the *Electronic Vision(s)* group for their friendliness and for the nice atmosphere.

Grazyna Gorny (University of Lethbridge, Canada) for allowing me to print her photograph of a stained neuron.

All who helped on the correction of this thesis.

Bettina Ehrlich for being around when things got stressful, for making even these times nice times.