RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

Roman Martel

Generative Properties of LIF-based Boltzmann Machines

Master thesis

HD-KIP-15-86

KIRCHHOFF-INSTITUT FÜR PHYSIK

Department of Physics and Astronomy
University of Heidelberg

Master Thesis
in Physics
submitted by
Roman Martel
born in Tokmok, Kyrgyzstan

September 2015

Generative Properties of LIF-based Boltzmann Machines

This Master Thesis has been carried out by Roman Martel at the
Kirchhoff Institute for Physics
Ruprecht-Karls-Universität Heidelberg
under the supervision of
Prof. Dr. Karlheinz Meier

Generative Properties of LIF-based Boltzmann Machines

Boltzmann machines are artificial neural networks which are able to learn generative models of real world data. Variations of Boltzmann machines are currently successfully applied to many machine learning tasks. The generation of samples (sampling) from a learned model in Boltzmann machines is computationally expensive but, in principle, massively parallel, which makes it appealing for an implementation on neuromorphic computing platforms. This has motivated the development of Boltzmann machines based on the leaky integrate-and-fire (LIF) neuron model, which is often implemented on neuromorphic devices. This thesis investigates the sampling properties of LIF-based Boltzmann machines compared to the one of classical Boltzmann machines. We observe differences in sampling, especially with regard to the ability to mix between several modes of the underlying model. Depending on the synapse model, connecting the neurons, an improvement or a decline of the mixing ability compared to classical sampling is found. Moreover, we present a mechanism based on short-term synaptic plasticity to improve the mixing ability for LIF-based Boltzmann machines. Using this mechanism, a better mixing ability compared to classical sampling for both applied synapse models is achieved.

Generative Eigenschaften LIF basierter Boltzmann Maschinen

Boltzmann-Maschinen sind künstliche neuronale Netze, die in der Lage sind generative Modelle realistischer Daten zu lernen. Variationen von Boltzmann-Maschinen werden derzeit erfolgreich in vielen Problemstellungen des maschinellen Lernens angewendet. Die Erzeugung von Stichproben eines gelernten Modells in Boltzmann-Maschinen ist rechenintensiv aber im Prinzip massiv parallelisierbar, was sie attraktiv für eine Implementierung auf neuromorphen Plattformen macht. Dies hat die Entwicklung von Boltzmann-Maschinen motiviert, die auf dem leaky integrate-and-fire (LIF) Neuronenmodell basieren, welches häufig auf neuromorphen Plattformen implementiert ist. Diese Arbeit untersucht die Eigenschaften LIF-basierter Boltzmann-Maschinen bei der Erzeugung von Stichproben im Vergleich zu klassischen Boltzmann Maschinen. Wir beobachten dabei Differenzen insbesondere im Hinblick auf die Fähigkeit zwischen verschiedenen Moden des zugrunde liegenden Modells zu wechseln. In Abhängigkeit vom Synapsenmodell, welches die Neuronen verbindet, finden wir eine Verbesserung oder einen Rückgang dieser Mixfähigkeit im Vergleich zur klassischen Erzeugung von Stichproben. Darüber hinaus präsentieren wir einen Mechanismus zur Verbesserung der Mixfähigkeit LIF-basierter Boltzmann-Maschinen, der auf synaptischer Kurzzeitplastizität basiert. Mit diesem Mechanismus wird für beide angewendeten Synapsenmodelle eine bessere Mixfähigkeit im Vergleich zur klassischen Erzeugung von Stichproben erreicht.

Contents

1. Introduction

The mammalian neocortex offers an, as far as known, unmatched performance in the analysis of noisy and ambiguous data (i.e. real world data), while having a power consumption of only 10-20 watts (*Javed et al.*, 2010). This inspired the development of artificial neural network models as a branch of machine learning. The *Boltzmann machine* (*Ackley et al.*, 1985) is an example of such a model, which is able to learn a generative model of real world data. Variations of Boltzmann machines are applied to many tasks including image classification (*Schmah et al.*, 2008; *Larochelle and Bengio*, 2008), image recognition and denoising (*Tang et al.*, 2012), modeling motion patterns (*Taylor et al.*, 2006; *Taylor and Hinton*, 2009) and acoustic modeling (*Mohamed and Hinton*, 2010). However, learning and stochastic inference via sampling for large Boltzmann machines is computationally expensive on conventional computing architectures, but, in principle, it can be made massively parallel (*Ackley et al.*, 1985). It can therefore take full advantage of computing architectures which are itself inspired by the massively parallel structure of the neocortex. One example for this so-called *neuromorphic hardware* is the HICANN (High Input Count Analog Neural Network), which is developed in the Electronic Visions group at the Heidelberg University. It represents a physical implementation of a network of spiking neuron models. However, several steps need to be taken to map the learning and sampling processes of a classical Boltzmann machine to such networks. In *Petrovici et al.* (2013) an approach was presented to perform sampling in networks of spiking neurons. The underlying neuron model of this approach is the *leaky integrate-and-fire* (LIF) model (*Lapicque*, 1907). These networks are therefore called LIF-based Boltzmann machines. Building up on this approach, it could further be demonstrated that learning can also be performed in LIF-based Boltzmann machines (*Leng*, 2014; *Weilbach*, 2015). This thesis aims to investigate the properties of the sampling process in LIF-based Boltzmann machines with a focus on mixing issues. These issues have already been reported for classical Boltzmann machines (*Hinton et al.*, 2004; *Salakhutdinov*, 2009) and prevent the sampling process to reproduce the full model of a Boltzmann machine. For this purpose, different approaches to visualize the sampling results for classical and LIF-based Boltzmann machines will be tested and compared. This will be done for several examples, including some which are deliberately designed to reveal mixing issues. Furthermore, an approach to solve these mixing issues in LIF-based Boltzmann machines, using *short-term plasticity*, will be applied. This approach will be compared with *adaptive simulated tempering* (AST) (*Salakhutdinov*, 2010), which is a method that addresses the same issue in classical Boltzmann machines.

Outline

In this thesis we used software tools to simulate the networks of spiking neuron models. Chapter 2 therefore introduces the software, which we have applied. However, as our final goal is to use the neuromorphic hardware to emulate these networks, it will also describe HICANN and its predecessor Spikey. Chapter 3 describes the theoretical models and tools used in this work. This includes Boltzmann machines, sampling, neuron and synapse models, the short-term plasticity model and visualization techniques. These techniques are employed in Chapter 4 to visualize the behavior of sampling in LIF-based and classical Boltzmann machines for several examples. In Chapter 5 we apply short-term plasticity to mitigate the mixing problem and use the visualization techniques to demonstrate the results. Finally Chapter 6 and Chapter 7 provide discussion and outlook for the results of this work.

2. Simulators and Emulators of Neural Networks

In this thesis we will investigate sampling in classical and LIF-based Boltzmann machines (BMs). For classical BMs (Section 3.2) we use standard sampling algorithms (Section 3.3), which we have implemented on our own. Sampling in LIF-based BMs (Section 3.8), however, is based on modeling networks of spiking neurons. The traditional approach to model such networks was using software frameworks to simulate them on general-purpose computers or high performance computing clusters. An alternative approach is to use hardware architectures which are itself inspired by the structure of neural networks. Today there are several approaches to develop such hardware architectures, which led to the *neuromorphic computing* subfield of neuroscience. In order to differentiate more easily between software and hardware implementations we use the term simulation when referring to spiking neural network simulations in software and emulation for hardware.
Currently there exists a large variety of software simulators and neuromorphic hardware devices. Interfacing with these systems becomes a critical issue, as models developed on one system typically do not run on others. Therefore, we start this chapter by presenting the PyNN software framework, which aims to mitigate this issue and which is also widely used throughout this thesis. Afterwards, we consider the software frameworks which do the actual simulation. Here we will just mention the software in detail that is used within this thesis. Finally, we briefly introduce neuromorphic hardware architectures. In this thesis no emulations on hardware were performed, but, as it is our aim to eventually run LIF-based BMs on hardware as well, an overview of the typical models and features supported by the hardware systems is indispensable to understand certain design choices of our approach.

2.1. Interfacing with Simulators and Emulators - PyNN

The diversity of software simulators gives modelers the option to choose simulators which are adjusted to their specific needs. However, it decreases the reproducibility of experiments (*Davison et al.*, 2008). Often, large parts of code for a model have to be rewritten to switch the simulation environment. To address this problem PyNN (pronounced 'pine') has been developed (*Davison et al.*, 2008). It is a modeling language for neural networks which serves as a front-end for several simulators. It can, however, also serve as a front-end for some hardware emulators. It uses the Python APIs, such as PyNEST (Section 2.2.1), which many simulation and emulation platforms provide, to achieve this. An overview of the supported simulation and emulation back-ends is shown in Fig. 2.1.

The user can create a whole network model completely in PyNN and choose his desired back-end afterwards. This also extremely simplifies the transition from standard simulators to a hardware emulator, which is our final goal.

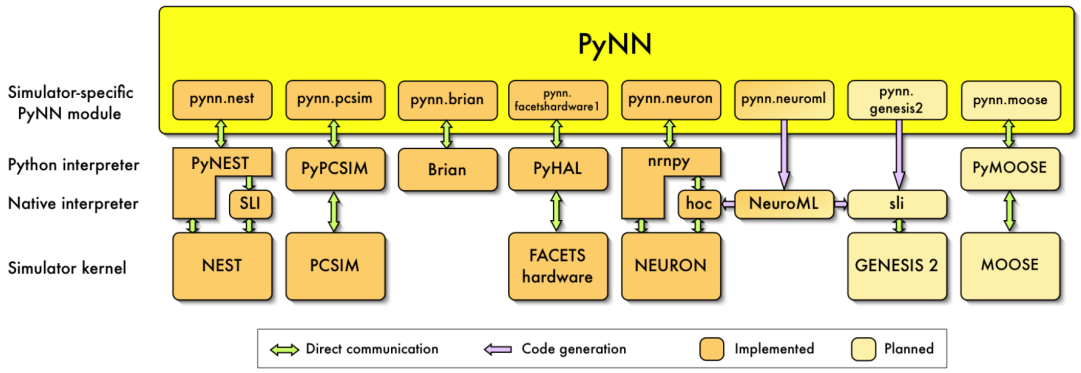


Figure 2.1.: Schematic of the simulator-agnostic modeling language PyNN. It can serve as a front-end for software simulators like NEST (*Diesmann and Gewaltig*, 2002), NEURON (*Hines and Carnevale*, 2003), PCSIM (*Pecevski et al.*, 2009), Brian (*Goodman and Brette*, 2008), NeuroML (*Gleeson et al.*, 2010), GENESIS (*Bower and Beeman*, 1998) and MOOSE (*Ray and Bhalla*, 2008). Furthermore, it supports the usage of the neuromorphic hardware developed within the FACETS and BrainScaleS projects (*Brüderle et al.*, 2011). Taken from *Brüderle et al.* (2011).

2.2. Simulation Software

A lot of simulation software exists in the field of computational neuroscience (*Brette et al.*, 2007). They make it possible to create network models on a high level using *populations* of neurons and *projections* between them. Many of them can handle more abstract neuron models such as the leaky integrate-and-fire model (Section 3.5) which we will use throughout this work. The intended simulation hardware devices are general purpose computers or supercomputers. We will explicitly just mention those frameworks that are used within this work.

2.2.1. NEST

The focus of the NEural Simulation Tool (NEST) (*Diesmann and Gewaltig*, 2002) is the simulation of large networks of point neurons with biologically realistic connectivity patterns. It is designed to be run on supercomputers and is therefore optimized for high performance computing. Hence it easily copes with simulations including 10^5 neurons assuming a realistic connectivity (*Morrison et al.*, 2005).
NEST features several user interfaces, one of which is PyNEST. It is an Python API that translates high level instructions, like creating or connecting populations of neurons, into

the simulation language SLI. SLI is a stack-based language derived from PostScript (*Inc.*, 1999) and serves as the native interface to NEST.
With PyNEST it is possible to interface with NEST using the PyNN front-end.

2.2.2. SBS

SBS (spike-based sampling) is a Python package which implements stochastic leaky integrate-and-fire (LIF) sampling (Section 3.8), which is used in LIF-based BMs. It is described in *Breitwieser* (2015). SBS operates on top of PyNN and NEST and uses the Python package numpy (*Numpy*, 2012) and matplotlib (*Hunter*, 2007) for plotting. It takes care of the parameter translation between abstract and LIF regime (calibration) for given neuron parameters as described in Section 3.8 and allows the evaluation of Boltzmann distributions in LIF networks with constant weights and biases. SBS is used throughout this work for all results concerning LIF sampling.

2.3. Neuromorphic Hardware

Neuromorphic hardware tries to realize the function of biological neural systems by emulating their structure. The first successful examples were neuromorphic sensors such as silicon retinas (*Lichtsteiner et al.*, 2008) or cochleas (*Liu and Delbruck*, 2010). Meanwhile also several neuromorphic computing platforms have been developed like HICANN (*Schemmel et al.*, 2008), SpiNNaker (*Furber et al.*, 2014), TrueNorth (*Merolla et al.*, 2014), Neurogrid (*Benjamin et al.*, 2014), Miniatur (*Neil and Liu*, 2014), IFAT (*Vogelstein et al.*, 2007) and NeuroDyn (*Yu and Cauwenberghs*, 2009). A review of many projects can be found in *Cattell and Parker* (2012). Many of these approaches are completely digital, which means that they numerically calculate the dynamics of the neuron models. SpiNNaker, for example, consists of a network of general purpose ARM986 processors, which are also used for smartphones, digital cameras and several embedded systems. The key innovation here is, according to *Furber et al.* (2014), the communication infrastructure which is optimized to carry very large numbers of very small packets containing spike events.
The TrueNorth architecture, as another example, uses instead an ASIC (application-specific integrated circuit) to implement digital processors which are specialized to numerically integrate the differential equations of the neuron models.
In this thesis, however, we especially aim to use architectures of a particular branch of neuromorphic hardware, which is often called *physical modeling*. It has its origins in *Mead and Mahowald* (1988), *Mead* (1990) and *Mead* (1989). The idea in these architectures is to implement physical representations of neuron models on hardware, such that their dynamics can be observed instead of numerically calculated. Compared to software simulations, this has the advantage that one looses the computational overhead necessary for general-purpose CPUs, which makes the architectures more efficient in terms of power consumption. Furthermore the architectures are inherently parallel which facilitates scaling of the emulated network sizes. This makes hardware emulations for large systems faster than software simulations.

In the following, we will describe two hardware devices of this physical modeling branch in more detail which come into consideration to serve as a platform for LIF-based BMs.

2.3.1. Spikey

The Spikey chip was developed within the FACETS project (Fast Analog Computing with Emergent Transient States) which is a consortium of 15 groups in 7 European countries (*FACETS*, 2010). A description for a modeler's point of view can already be found in *Petrovici* (2015). We will therefore give just a short summary here.
The Spikey chip uses analog very-large-scale integration (VLSI) circuits to physically model neurons and synapses. As a consequence, quantities like the membrane potential evolve continuously in contrast to implementations on digital hardware. The implemented neuron model on Spikey is a VLSI version of the leaky integrate-and-fire (LIF) model (Section 3.5). It uses exponential-shaped conductance-based synapses (Section 3.6). In total the chip contains 384 neurons and 128K synapses. In order to achieve these numbers of neurons and synapses on the restricted chip size, electrical components, like the capacity modeling the membrane capacity, with a small size are used. This leads to shorter time constants for the neuron dynamics. As a consequence, the neuron dynamics on Spikey run with a speedup factor of 10^4 compared to biological real-time. The parameters of the implemented neurons and synapses can be varied, which enables the emulation of different types of neurons and synapses. The synapse weights can be set with a 4-bit resolution.
Short-term plasticity (STP) to modulate the strength of the synaptic activation is available on Spikey (*Schemmel et al.*, 2007). It is similar to the model by Tsodyks and Markram (*Tsodyks and Markram*, 1997). This model uses one more variable than the one we will use in this thesis (Section 3.7), but by changing certain parameters it can have the identical behavior. However, the Spikey STP implementation can only be either facilitating or depressing and not both at the same time. In this work we will just use depressing STP but a combination of both could be a promising extension of our approach.
The propagation of spikes is a combination of analog on-chip communication and digital off-chip communication. In off-chip communication the spikes are represented as time-stamped events, which is described in more detail in *Schemmel et al.* (2006).
The Spikey system supports PyNN as an API.

2.3.2. HICANN

The HICANN (High Input Count Analog Neural Network) chip (*Schemmel et al.*, 2008, 2010) was developed within the BrainscaleS project (*BrainScaleS*, 2012), which started in 2011. It is the follow-on project to FACETS. Further versions of the HICANN chip are currently still under development within the Human Brain Project (*Markram*, 2012). Detailed descriptions for a modeler's point of view can be found in *Petrovici* (2015); *Probst* (2014); *Breitwieser* (2015).
To go from the Spikey chip to the HICANN architecture the concept of *wafer-scale in-*

tegration was used. The motivation behind this was that, for the same reasons as in Spikey, the HICANN chip has fast neurons with a speed-up factor of approximately 10^4 compared to biological real-time. This makes a high bandwidth necessary to manage the communication of generated pulse-events between several chips. As communication between large distances is energetically expensive and slow, it is advantageous to reduce the distance between the chips as much as possible. The approach in wafer-scale integration is therefore, instead of producing a wafer of chips and then cutting out each single chip, as it was done for Spikey, one uses the whole wafer as one neuromorphic substrate. The wafer contains 384 HICANN chips. The HICANN chips are grouped in so-called *reticles* on the wafer. These are the regions which can be simultaneously exposed to photolitography during the fabrication. In a post-processing step connections between all HICANN chips are created on the wafer. This enables a high bandwidth for on-wafer pulse-event communication (*Schemmel et al.*, 2008). Horizontal and vertical channels connect the HICANN chips between and within the reticles (see Fig. 2.2).
The neurons emulate the dynamics of the adaptive-exponential integrate-and-fire model (AdEx) (*Brette and Gerstner*, 2005). It is possible, though, to effectively disconnect both the circuit for the adaption mechanism and the exponential term from the membrane capacitance, such that the LIF model, which is exclusively used in this thesis, can also be emulated. As in Spikey, exponential-shaped conductance-based synapses are used.
A single HICANN chip contains 512 neurons and $224 \cdot 512$ synapses. Hence, there are in total 196,608 neurons and approximately 44 million synapses on each wafer. As in Spikey, each synapse has a 4-bit digital weight. However, two synapses of adjacent rows can be combined which leads to a weight resolution of 8 bits.
The implemented short-term plasticity mechanism is motivated by *Markram et al.* (1998). We will consider this mechanism later in this thesis (Section 3.7). However, as described in *Billaudelle* (2014), in contrast to the theoretical model, there is a linear recovery of the maximal height of post-synaptic potentials (PSPs) after a spike burst. Like in Spikey, the hardware implementation does not allow simultaneous depression and facilitation, which is in contrast to the original plasticity mechanism. Furthermore, an STDP mechanism is implemented in the synapses. Details to this can be found in *Brüderle et al.* (2011) and *Schemmel et al.* (2007).
The scheme of the HICANN with its communication layers is illustrated in Fig. 2.2. The terminals for the off-wafer communication are represented by custom digital ASICs. They are called *Digital Network Chips* (DNCs) and are backed by a field programmable gate array (FPGA) design that handles the packet routing (*Hartmann et al.*, 2010). In total 44 billion events per second can be communicated between the wafers. This makes it in principle possible to connect many wafers to emulate large-scale neural networks on hardware.
As in Spikey, one can use PyNN as an API to define models which shall run on HICANN. This may in future allow a relative simple transition from the software simulations in this thesis to HICANN. The translation of the PyNN models to the configuration of the neuromorphic device is described in *Brüderle et al.* (2011) and *Ehrlich et al.* (2010). There has already been an attempt to run a PyNN model of LIF-based BMs on the HICANN chip (*Leng*, 2014). The conclusion was that due to some technical problems the

implementation of BMs on the chip is currently not possible and should be postponed until the next generation of HICANN chips is available.

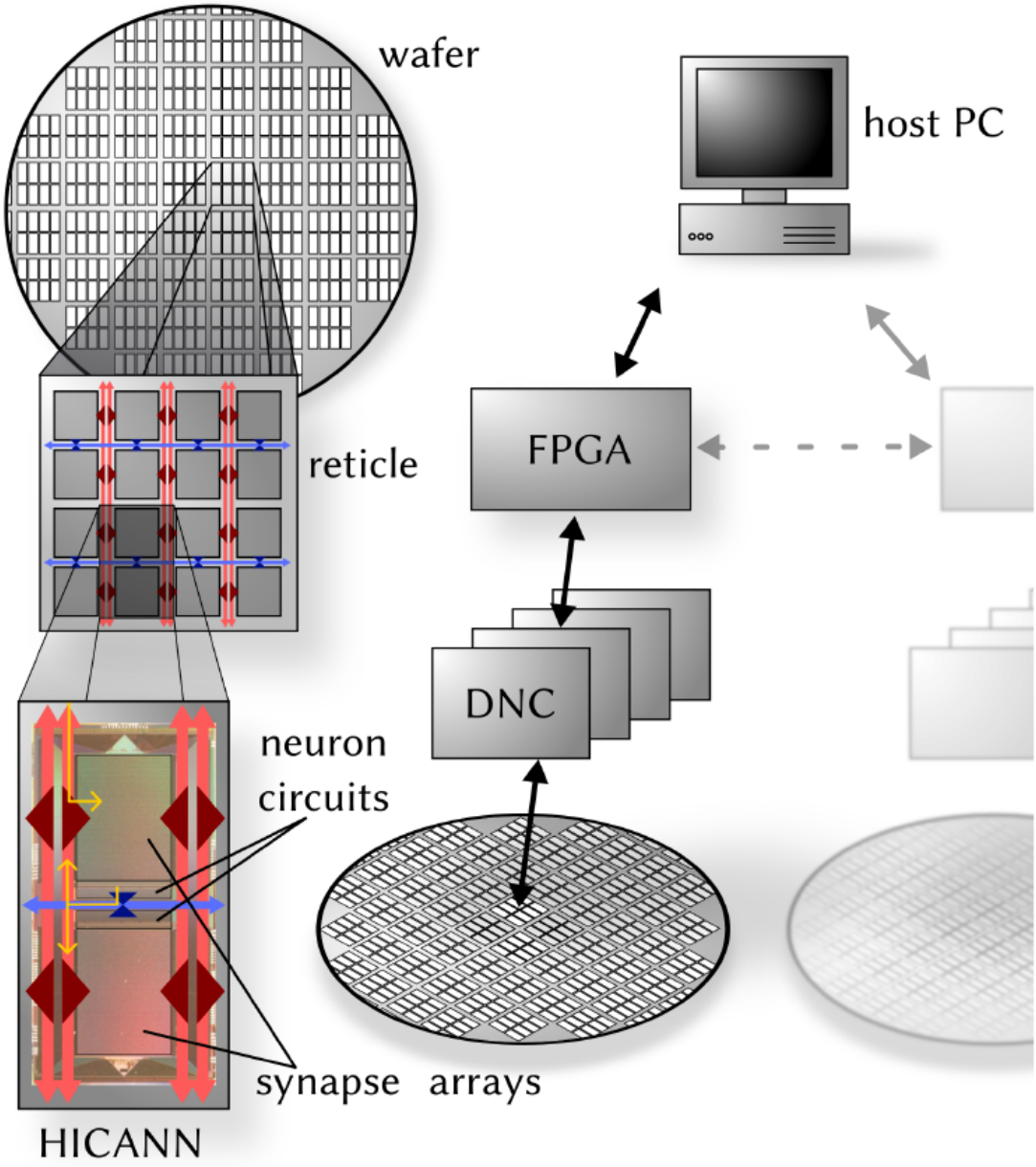


Figure 2.2.: HICANN scheme with the communication layers. See text for details.

3. Theoretical Background

In this chapter we will introduce the theoretical methods and models which will be used throughout this thesis.

3.1. Kullback-Leibler Divergence

The *Kullback-Leibler divergence* D_{KL} is used within this thesis mainly as a difference measure between two probability distributions. It is defined as

$$D_{KL}(p||q) = \sum_{\mathbf{z}\in S} p(\mathbf{z}) \ln \left[\frac{p(\mathbf{z})}{q(\mathbf{z})}\right] \quad , \tag{3.1}$$

where $S = \{\mathbf{z}_1, \mathbf{z}_2, \dots\}$ denotes the state space for the probability distributions $p(\mathbf{z})$ and $q(\mathbf{z})$. In *Bishop* (2009) an interpretation of the D_{KL} motivated by information theory is given. It is easy to show that $D_{KL}(p||q) \geq 0$:

$$\begin{aligned}
D_{KL}(p||q) &= \sum_{\mathbf{z}\in S} p(\mathbf{z}) \ln \left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right) && (3.2)\\
&= \sum_{\{\mathbf{z}\in S|p(\mathbf{z})>0\}} p(\mathbf{z}) \ln \left(\frac{p(\mathbf{z})}{q(\mathbf{z})}\right) && (3.3)\\
&= - \sum_{\{\mathbf{z}\in S|p(\mathbf{z})>0\}} p(\mathbf{z}) \ln \left(\frac{q(\mathbf{z})}{p(\mathbf{z})}\right) && (3.4)\\
&\geq - \sum_{\{\mathbf{z}\in S|p(\mathbf{z})>0\}} p(\mathbf{z}) \left(\frac{q(\mathbf{z})}{p(\mathbf{z})} - 1\right) && (3.5)\\
&= \underbrace{\sum_{\{\mathbf{z}\in S|p(\mathbf{z})>0\}} p(\mathbf{z})}_{=1} - \underbrace{\sum_{\{\mathbf{z}\in S|p(\mathbf{z})>0\}} q(\mathbf{z})}_{\leq 1} && (3.6)\\
&\geq 0 && (3.7)
\end{aligned}$$

where in (3.5) we used that

$$\begin{aligned}
\ln(x) &\leq x - 1 && (3.8)\\
\Leftrightarrow \quad -\ln(x) &\geq -(x-1) && (3.9)
\end{aligned}$$

We note that this proof holds just in case that we compute the D_{KL} for the whole state space S. Later in this thesis we will use a "reduced D_{KL}" where we sum just over a subspace of the whole state space (Section 4.2.2). In this situation a $D_{KL} < 0$ becomes possible and has to be taken into account.

3.2. Boltzmann Machine

A Boltzmann Machine (BM) (*Hinton*, 2007; *Ackley et al.*, 1985) is a physical implementation of a set of binary random variables (RVs) that follows a Boltzmann distribution. The RVs are represented by binary units which are symmetrically connected in a network. In the standard BM every unit is connected with every other one. This is illustrated in Fig. 3.1. In this general form the Boltzmann distribution assigns each state $\mathbf{z}$ of the vector over all K binary RVs $\mathbf{Z} = (Z_1, \ldots, Z_K)$ a probability according to

$$p(\mathbf{z}) = \frac{1}{Z} \exp\left[-E(\mathbf{z})\right] \quad . \tag{3.10}$$

The constant Z is called the *partition function* and ensures that the probability distribution is normalized to 1:

$$Z = \sum_{\mathbf{z}} \exp\left[-E(\mathbf{z})\right] \quad . \tag{3.11}$$

The term $E(\mathbf{z})$ is called the *energy function*. It determines the energy of a state

$$E(\mathbf{z}) = -\sum_{i,j} \frac{1}{2} W_{ij} z_i z_j - \sum_i b_i z_i \quad , \tag{3.12}$$

with arbitrary real-valued connection weights W_{ij} and biases b_i. The weights are symmetric i.e. $W_{ij} = W_{ji}$. This leads to the factor of $\frac{1}{2}$ because when we sum over all combinations of i, j we count every weight connection twice. From (3.10) it follows, as one would expect in a physical system, that the states with lowest energy have the highest probability. BMs are used in machine learning to learn internal representations of data which is shown to the machine. They are consequently generative models. However, as there is no conceptional difference between data and labels, it is possible to use BMs as a discriminative model as well. To achieve this one can learn a generative representation of the data together with the labels. To do classification one then performs a pattern completion task where the data is shown to the system and the label completed.
During the learning process the weights and biases of the BM are adjusted such that its represented probability distribution fits the training data as well as possible. For fully connected Boltzmann machines learning is too computationally demanding to be useful for any practical application in machine learning (*Hinton*, 2007). However, it can be simplified by imposing restrictions on the network topology. This leads us to restricted Boltzmann machines (RBMs) described in the next section.

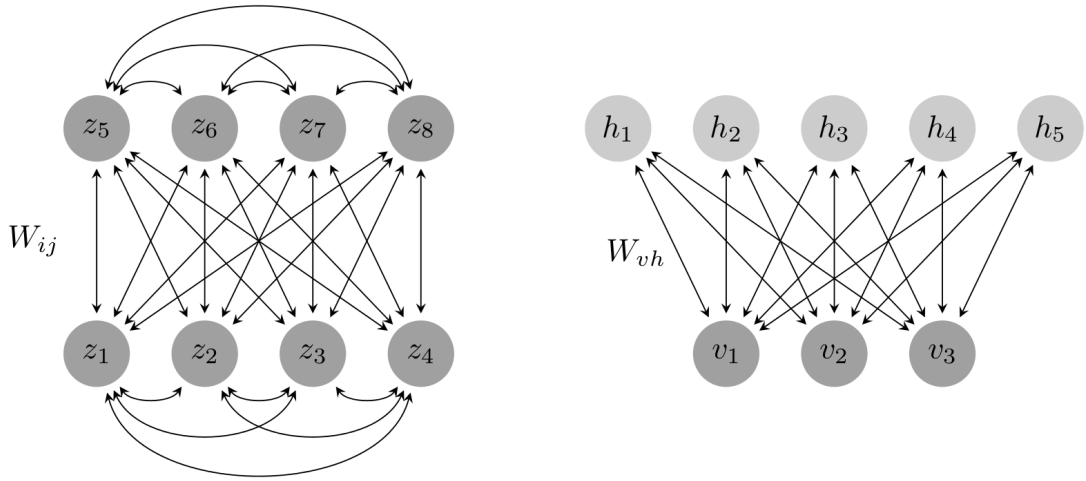


Figure 3.1.: Schematic of the network model of a fully connected Boltzmann machine (**left**) and of a restricted Boltzmann machine (**right**). In an RBM, the units form two groups, a so-called visible and a hidden layer. The connections are restricted to be only between the layers but not within them.

3.2.1. Restricted Boltzmann Machine

The restricted Boltzmann machine (RBM) (*Smolensky*, 1986) is a special case of the general BM. Its underlying graphical model is bipartite which is illustrated in Fig. 3.1. The two parts are called visible and hidden layer. Connections in this network topology are restricted to be between the layers and not within them. The probability distribution for the states which are now partitioned into a visible $\mathbf{v}$ and a hidden part $\mathbf{h}$ can be deduced from (3.10), (3.12)

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp\left[-E(\mathbf{v}, \mathbf{h})\right] \quad , \tag{3.13}$$

with the *energy function*

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \text{visible}} a_i \, v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{i,j} W_{ij} \, v_i h_j \quad . \tag{3.14}$$

In the last years, models which are based on RBMs have been shown to be very useful for many kinds of applications in machine learning. Examples are image recognition and denoising (*Tang et al.*, 2012), learning generative models of images (*Le Roux et al.*, 2011), image classification (*Schmah et al.*, 2008; *Larochelle and Bengio*, 2008), modeling motion patterns (*Taylor et al.*, 2006; *Taylor and Hinton*, 2009), acoustic modeling (*Mohamed and Hinton*, 2010), movie recommendations (*Salakhutdinov et al.*, 2007) or dimensionality reduction (*Hinton and Salakhutdinov*, 2006). Several RBMs can be stacked together to form so-called deep Boltzmann machines (*Salakhutdinov and Hinton*, 2009), which is a

widely used model as well and can also be implemented with spiking neurons (*Leng*, 2014).
One important reason for the success of RBMs is that there exist effective training mechanisms which we will regard later (Section 3.9). The restricted structure also accelerates the extraction of information from the model via sampling (see Section 3.3).
Another important feature of RBMs, and in general for every BM with hidden units, is that the training data is usually just represented by a part of the model, namely the visible layer. The hidden layer serves as a feature extractor. The model of the data is then the Boltzmann distribution over all units in the machine marginalized over the hidden units

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} \exp\left[-E(\mathbf{v}, \mathbf{h})\right] \quad . \tag{3.15}$$

Hence, although the state of the whole machine follows a Boltzmann distribution, the distribution represented by the visible layer is, in principle, not restricted to any particular form. This makes it possible to learn good models for data sets which are not necessarily Boltzmann distributed. These are the reasons why we will mainly consider RBMs in this thesis.
The features for which the hidden units code for can often be visualized by looking at their *receptive fields*. These are pictures obtained from the weight connections of the hidden units to the visible layer. If, for example, the visible layer represents 12x12 pixel images in a data set, we can arrange the weights of each hidden unit in the same order to obtain 12x12 pixel images as well. These images represent the receptive fields of the hidden units. An example for them can be seen in Fig. 4.21a.
The particular shape of the marginal distributions over the visible and hidden layer is easy to calculate, as they just involve information from the other layer, due to the missing connections within the layers. We will make use of this later, when we visualize the sampling performance of large RBMs (Chapter 4). Therefore, we will give here the analytic derivation for the marginal distribution of the visible layer of an RBM with N visible and M hidden units

$$
\begin{aligned}
p(\mathbf{v}) &= \frac{1}{Z}\sum_{\mathbf{h}} \exp\left[-E(\mathbf{v},\mathbf{h})\right] \\
&= \frac{1}{Z}\sum_{\mathbf{h}} \exp\left(\sum_{i=1}^{N} a_i\, v_i + \sum_{j=1}^{M} b_j h_j + \sum_{i,j} W_{ij}\, v_i h_j\right) \\
&= \frac{1}{Z}\exp\left(\sum_{i=1}^{N} a_i v_i\right)\sum_{\mathbf{h}} \exp\left[\sum_{j=1}^{M} h_j\Big(b_j + \sum_{i=1}^{N} W_{ij}\, v_i\Big)\right] \\
&= \frac{1}{Z}\exp\left(\sum_{i=1}^{N} a_i v_i\right)\sum_{h_1}\cdots\sum_{h_M}\prod_{j=1}^{M} \exp\left[h_j\Big(b_j + \sum_{i=1}^{N} W_{ij}\, v_i\Big)\right] \\
&= \frac{1}{Z}\exp\left(\sum_{i=1}^{N} a_i v_i\right)\sum_{h_1} \exp\left[h_1\Big(b_1 + \sum_{i=1}^{N} W_{i1} v_i\Big)\right]\cdot\ldots\cdot\sum_{h_M} \exp\left[h_M\Big(b_M + \sum_{i=1}^{N} W_{iM} v_i\Big)\right] \\
&= \frac{1}{Z}\exp\left(\sum_{i=1}^{N} a_i v_i\right)\prod_{j=1}^{M}\left\{\sum_{h_j\in\{0,1\}} \exp\left[h_j\Big(b_j + \sum_{i=1}^{N} W_{ij} v_i\Big)\right]\right\} \\
&= \frac{1}{Z}\exp\left(\sum_{i=1}^{N} a_i v_i\right)\prod_{j=1}^{M}\left[1 + \exp\Big(b_j + \sum_{i=1}^{N} W_{ij} v_i\Big)\right] \quad .
\end{aligned}
$$

The computational complexity of evaluating the final expression grows linearly with the number M of hidden units while the initial sum had an exponential dependency. This drastic improvement was possible because we utilized the independence of the hidden units from each other. This allows it to evaluate marginal visible or hidden distributions, despite the fact that the other layer might be large.
One can similarly show the shape of the marginal hidden distribution to be

$$
p(\mathbf{h}) = \frac{1}{Z}\exp\left(\sum_{j=1}^{M} b_j h_j\right)\prod_{i=1}^{N}\left[1 + \exp\Big(a_i + \sum_{j=1}^{M} W_{ij} h_j\Big)\right] \quad . \tag{3.16}
$$

3.3. Sampling

Exact inference becomes difficult for high-dimensional probability distributions, because the possible state space grows exponentially. A Boltzmann machine with N units has for example 2^N possible states. Sampling is a method used to approximate inference (*Bishop*, 2009). In this section we will introduce several sampling approaches, which we have also used in this thesis.

3.3.1. Metropolis-Hastings Algorithm

The *Metropolis-Hastings* algorithm (*Hastings*, 1970) is a *Markov chain Monte Carlo* (MCMC) sampling method (*Metropolis and Ulam*, 1949). Here the sequence of samples

$\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(\tau)}$ forms a Markov chain. One usually uses first-order Markov chains in which the probability of the state at a certain time step $\mathbf{z}^{(\tau)}$ only depends on the previous state

$$p(\mathbf{z}^{(\tau)}|\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(\tau-1)}) = p(\mathbf{z}^{(\tau)}|\mathbf{z}^{(\tau-1)}) \quad . \tag{3.17}$$

The actual sampling process consists of two parts. First a new state $\mathbf{z}^*$ is proposed according to a proposal distribution $q(\mathbf{z}^*|\mathbf{z}^{(\tau)})$ from which it is easier to draw samples. Then this state is accepted with an acceptance probability

$$A(\mathbf{z}^*, \mathbf{z}^{(\tau)}) = \min\left(1, \frac{q(\mathbf{z}^{(\tau)}|\mathbf{z}^*)\tilde{p}(\mathbf{z}^*)}{q(\mathbf{z}^*|\mathbf{z}^{(\tau)})\tilde{p}(\mathbf{z}^{(\tau)})}\right) \quad , \tag{3.18}$$

with $\tilde{p}(\mathbf{z}^{(\tau)}) = Z \cdot p(\mathbf{z}^{(\tau)})$ denoting the unnormalized probability distribution, which is used because the partition functions cancel each other. If the proposed sample $\mathbf{z}^*$ is accepted, we set $\mathbf{z}^{(\tau+1)} = \mathbf{z}^*$, otherwise it is discarded and $\mathbf{z}^{(\tau+1)} = \mathbf{z}^{(\tau)}$.
The fact that we don't need the expensive partition function anymore to generate samples makes Metropolis-Hastings sampling practical.

3.3.2. Gibbs Sampling

Gibbs sampling (*Geman and Geman*, 1984) can be seen as a special case of Metropolis-Hastings sampling. Here we sample every component $z_k^{(\tau)}$ of the current state vector $\mathbf{z}^{(\tau)}$ separately according to its conditional distribution. The proposal distribution from Metropolis-Hastings then becomes

$$q(\mathbf{z}^*|\mathbf{z}^{(\tau)}) = p(z_k^*|\mathbf{z}_{\setminus k}^{(\tau)}) \quad . \tag{3.19}$$

Using this and $p(\mathbf{z}) = p(z_k|\mathbf{z}_{\setminus k})p(\mathbf{z}_{\setminus k})$, the acceptance probability becomes

$$\begin{aligned} A(\mathbf{z}^*, \mathbf{z}^{(\tau)}) &= \frac{q(\mathbf{z}^{(\tau)}|\mathbf{z}^*)\ p(\mathbf{z}^*)}{q(\mathbf{z}^*|\mathbf{z}^{(\tau)})\ p(\mathbf{z}^{(\tau)})} && (3.20)\\ &= \frac{p(z_k^{(\tau)}|\mathbf{z}_{\setminus k}^*)\ p(z_k^*|\mathbf{z}_{\setminus k}^*)p(\mathbf{z}_{\setminus k}^*)}{p(z_k^*|\mathbf{z}_{\setminus k}^{(\tau)})\ p(z_k^{(\tau)}|\mathbf{z}_{\setminus k}^{(\tau)})p(\mathbf{z}_{\setminus k}^{(\tau)})} && (3.21)\\ &= 1 \quad , && (3.22) \end{aligned}$$

where in the last step we used that only z_k is changed which means $\mathbf{z}_{\setminus k}^* = \mathbf{z}_{\setminus k}^{(\tau)}$. Thus we always accept the proposed state. One complete Gibbs step is performed when all components z_k of the state $\mathbf{z}$ are updated. The order in which we update the components is arbitrary. For simplicity, one usually chooses a fixed periodic order.

3.3.3. Adaptive Simulated Tempering

For an MCMC sampling method to be able to sample from a desired stationary distribution $p^*(\mathbf{z})$ the Markov chain needs to be *ergodic*. This means that independent of

the initial state $\mathbf{z}^{(0)}$ the sampler will visit all possible states with the correct (relative) probability in the limit of infinite sampling steps:

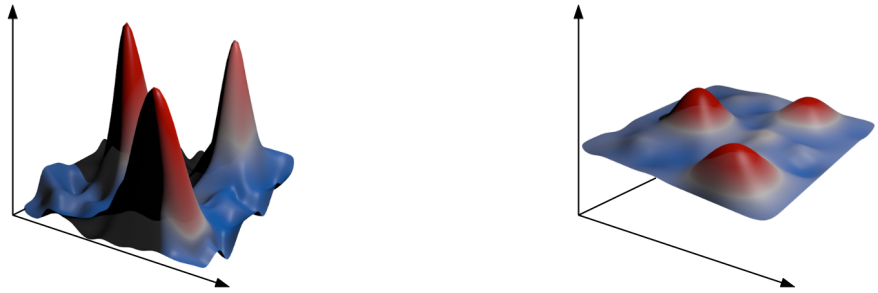
$$\lim_{\tau\to\infty} p(\mathbf{z}^{(\tau)}) = p^*(\mathbf{z}) \quad . \tag{3.23}$$

A necessary condition for ergodicity is *irreducibility*, which requires that as $\tau \to \infty$ it is possible to go from any state of the Markov chain to any other possible state in a finite number of steps with non-vanishing probability. In other words, every part of the state space should be accessible to the Markov chain.
Later in this thesis we will see cases in which this condition becomes critical (Section 4.2, Section 4.3.2). In these cases the irreducibility condition is theoretically still fulfilled, but in practice the probability to be in some areas of the state space is so low that the time that is typically needed to reach them vastly exceeds our currently possible simulation time. In these cases we speak of bad *mixing*, where mixing is loosely defined as the ability of the Markov chain to travel quickly through the state space. Here one can observe that some regions of the state space with a high probability mass become isolated, because they are surrounded by regions with a nearly vanishing probability. This is illustrated in Fig. 3.2a. It often happens that when using standard Gibbs sampling the Markov chain stays in one local mode with a high probability but does not find the other local modes. In the illustration this corresponds to being trapped in one of the three "mountains". This issue becomes especially important if one wants to take samples from large systems, because here the state space grows exponentially with the size of the system but typically the number of modes is still comparably small. An example for this would be a system which is trained on a high-dimensional structured data set like the MNIST data set of handwritten digits (*LeCun and Cortes*, 1998). We will investigate such systems later in this thesis (Section 4.3).

Adaptive Simulated Tempering (AST) (*Salakhutdinov*, 2010) represents one way to address this problem. Here, we will give just the intuition behind this method and refer for a more detailed description to *Leng* (2014).
The main idea behind AST is to introduce a temperature T_k into the probability distribution. Inspired by statistical mechanics we denote it in the Boltzmann distribution by an inverse temperature $\beta_k = \frac{1}{T_k}$

$$p(\mathbf{z}|k) = \frac{1}{Z_k} \exp\left[-\beta_k E(\mathbf{z})\right] \quad , \tag{3.24}$$

where k stands for the temperature levels, which are chosen to fulfill
$0 < \beta_K < \beta_{K-1} < \cdots < \beta_1 = 1$, with K being the total number of temperature levels. If we consider k itself to be a random variable, we sample from a combined state space of temperature and normal states. To do this one alternately takes first a new sample for the state $\mathbf{z}$ with Gibbs sampling (Section 3.3.2) and secondly a new sample for the temperature level k via Metropolis-Hastings (Section 3.3.1). The proposal distribution for Metropolis-Hastings for a new temperature level k^* given our current level $k^{(\tau)}$ is



(a) Rough probability landscape

(b) Flat probability landscape

Figure 3.2.: Illustration of different kinds of probability landscapes. The height and the color both stand for the probability of the region. Where a high position and red color means it is likely and a low position with blue color unlikely. One can imagine these landscapes to be a two-dimensional map of the N-dimensional distributions which we usually deal with. Figure 3.2a shows an example where mixing is important. There are three dominant modes which are surrounded by unlikely area. To switch between the modes becomes therefore very unlikely. Figure 3.2b shows an example where mixing is not critical.

chosen to be

$$q(k^*|k^{(\tau)}) = \begin{cases} \frac{1}{2}, & \text{if } k^* = k^{(\tau)} + 1 \ \vee \ k^* = k^{(\tau)} - 1 \\ 1, & \text{if } (k^* = 2 \wedge k^{(\tau)} = 1) \ \vee \ (k^* = K - 1 \wedge k^{(\tau)} = K) \\ 0, & \text{otherwise} \quad . \end{cases} \tag{3.25}$$

This approach alone is called *simulated tempering* (*Marinari and Parisi*, 1992). To make sure that the sampler spends equal time at every temperature level, one couples this approach with the Wang-Landau algorithm (*Wang and Landau*, 2001) to form AST. For every temperature level k, the Wang-Landau algorithm introduces additional factors g_k in the acceptance probability of Metropolis-Hastings ((3.18)), which yields

$$A(k^*, k^{(\tau)}) = \min\left(1, \frac{q(k^{(\tau)}|k^*)\tilde{p}(\mathbf{z}^{(\tau+1)}|k^*)g_{k^{(\tau)}}}{q(k^*|k^{(\tau)})\tilde{p}(\mathbf{z}^{(\tau+1)}|k^{(\tau)})g_{k^*}}\right) \quad . \tag{3.26}$$

These factors are increased every time we sample from the corresponding temperature level according to

$$g_k^{(\tau+1)} = g_k^{(\tau)} \ (1 + \gamma^{(\tau)} I(k^{(\tau+1)} \in \{k\})), \quad k = 1, \ldots, K \quad . \tag{3.27}$$

where I is the indicator function and $\gamma^{(\tau)} > 0$ the *weight adaption factor*. It can be shown that for $\gamma^{(\tau)} \to 0$ as $\tau \to \infty$ the Markov chain spends the same amount of time in each temperature level.

A typical example for the temperature evolution is shown in Fig. 3.3. When it goes into the high temperature regime the inverse temperature β becomes low. According to (3.24) this leads to a flattening of the distribution. Hence, when we start with a rough distribution like Fig. 3.2a, we obtain in the high temperature regime a distribution like Fig. 3.2b. Here mixing is simplified and the sampler can reach every state again. To obtain valid samples from the target distribution $p^*(\mathbf{z})$, we just count the samples taken from temperature level 1. As a consequence, if one chooses K temperature levels, it takes roughly K times longer to obtain the same number of samples as in Gibbs sampling. In Fig. A.4 one can see an image series taken from an animation which further illustrates this mechanism.

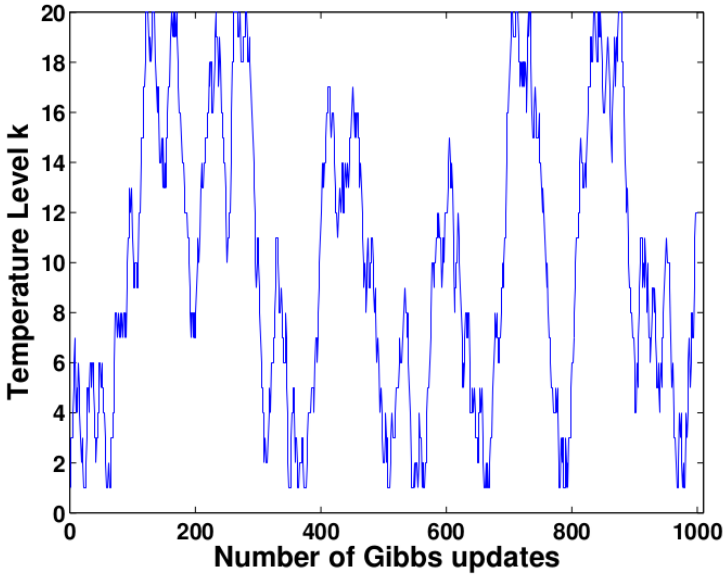


Figure 3.3.: Evolution of the temperature levels during AST. One alternately samples a new state using Gibbs sampling and a new temperature level using the Metropolis-Hastings algorithm. Only the Gibbs samples from temperature level 1 are taken as valid samples. Image taken from *Salakhutdinov* (2010).

3.4. Neural Sampling

The previously discussed methods for MCMC sampling are inconsistent with the dynamics of spiking neurons because they are reversible. This is incompatible with the refractory period of the neurons (see Section 3.5). With *Neural Sampling* (*Buesing et al.*, 2011) a different approach based on non-reversible Markov chains was taken, which tries to reflect inherent temporal processes of spiking neurons.
Each component z_k of the state $\mathbf{z}$ is encoded by a neuron ν_k such that

$$z_k(t) = 1 \;\Leftrightarrow\; \nu_k \text{ has fired within the time interval } (t-\tau,\, t\,] \quad ,$$

where τ represents the absolute refractory period of the neurons. One assumes that the membrane potential $u_k(t)$ of neuron ν_k at time t equals the log-odds of the corresponding state component z_k to be active given the state of all other units $\mathbf{z}_{\backslash \mathbf{k}}$

$$u_k(t) = \log \frac{p(z_k = 1|\mathbf{z}_{\backslash k})}{p(z_k = 0|\mathbf{z}_{\backslash k})} \quad . \tag{3.28}$$

For this assumption it is necessary that each neuron in the network is able to compute the right-hand side of (3.28). Therefore this equation is called *neural computability condition* (NCC). If we consider a Boltzmann distribution (3.10) with energy (3.12), we obtain for the NCC the membrane potential

$$u_k(t) = b_k + \sum_{i=1}^{K} W_{ik} z_i(t) \quad . \tag{3.29}$$

Resolving (3.28) for the probability of the neuron to be in the on-state yields

$$p(z_k = 1|\mathbf{z}_{\backslash k}) = \sigma(u_k(\mathbf{z}_{\backslash k})) := \frac{1}{1 + \exp[-u_k(\mathbf{z}_{\backslash k})]} \quad , \tag{3.30}$$

In other words, each neuron is required to have a logistic activation function. Due to the refractory period, the states $\mathbf{z}$ alone cannot represent a first-order Markov chain anymore. To obtain this property again so-called *internal state variables* ζ_k were introduced. Figure 3.4 illustrates the scheme of the transition operator operating on them. If the neuron ν_k spikes, its corresponding internal state is set to $\zeta_k = \tau$. Every following time step we reduce ζ_k by 1 until it reaches the value 1. Here it spikes or it is further reduced to 0. It then stays in the 0-state until it spikes again. The probability for neuron ν_k to spike can be calculated to be $\sigma(u_k - \log \tau)$.
The state of the neuron z_k is set to be 1 as long as $\zeta_k \geq 1$ otherwise the refractory period is considered to be over and the neuron switches to 0. Hence ζ_k acts like a counter for the last spike. It ensures that the information of the last state $(z_k^{(\tau-1)}, \zeta_k^{(\tau-1)})$ is sufficient to compute the current state. Thus, we have a first order Markov chain which simplifies the proof for its convergence towards a desired stationary distribution.
For details about this proof we refer to *Buesing et al.* (2011). For $\tau = 1$ this approach is equivalent to Gibbs sampling (Section 3.3.2).

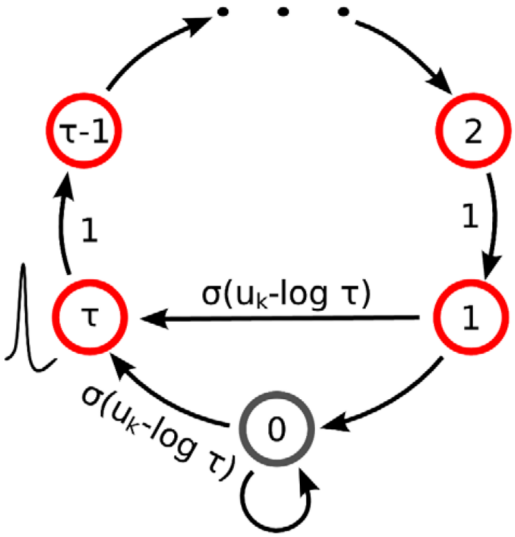


Figure 3.4.: Scheme of the transition operator for the internal state variables ζ_k. For the values $\zeta_k \in \{0, 1\}$ the neuron ν_k is able to spike with the probability $\sigma(u_k - \log \tau)$. If it spikes one sets $\zeta_k = \tau$. Afterwards it is decreased by 1 in every time step until it reaches the values 0 or 1. Here it is able to spike again which closes the cycle. Image taken from *Buesing et al.* (2011).

3.5. Leaky Integrate-and-Fire (LIF) Neuron Model

The leaky integrate-and-fire (LIF) model (*Lapicque*, 1907) is one of the simplest neuron models which still captures the most significant aspects of neural dynamics. It assumes neurons to be pointlike and contains just a one-dimensional neuron state space spanned by its membrane potential u, which obeys the ODE

$$C_m \frac{\mathrm{d}u}{\mathrm{d}t} = g_l(E_l - u) + I^{\mathrm{syn}} + I^{\mathrm{ext}} \quad , \tag{3.31}$$

with the membrane capacitance C_m, leak potential E_l, and leak conductance g_l. The input current I can be divided into two parts, a synaptic current I^{syn} which comes from the synaptic connections of the neuron to other neurons and an external current I^{ext} which allows further control over the model. Spikes are generally assumed to have nearly identical shape (*Gerstner and Kistler*, 2002; *Dayan and Abbott*, 2001). Under this assumption they transmit no additional information and just the spike time becomes important.
Therefore to avoid the necessity to model the complex shape of the spike one applies a threshold rule. Here a spike is emitted if the membrane potential u crosses a certain threshold θ from below. The spikes, however, consist just of the time of their occurrence and instead of performing an action potential the neuron's membrane potential is set to a reset potential ρ for the duration of the so-called refractory time τ_{ref}.

The LIF model represents a trade-off between functionality and computational complexity and is therefore used in many simulation and hardware projects. In spite of its simplicity, it is able to perform complex computational tasks, as demonstrated in LIF Sampling (Section 3.8).

3.6. Exponential- and Alpha-Shaped Synapse Model

In general synaptic interactions are modeled by interaction kernels which are linearly summed up over different synapses and time

$$f^{\text{syn}}(t) = \sum_{\text{syn}\,k} \sum_{\text{spk}\,s} w_k \epsilon_k(t - t_s) \quad , \tag{3.32}$$

where w_k denotes the weight and ϵ_k the interaction kernel of the synapse k and "spk" stands for spike. In an intuitive model one can describe the arrival and removal of neurotransmitters at the post-synaptic site with two exponential functions with time constants τ_{rise} and τ_{fall}

$$\epsilon(t) = A\Theta(t)\frac{1}{\tau_{\text{rise}} - \tau_{\text{fall}}}\left(\exp\Big(-\frac{t}{\tau_{\text{rise}}}\Big) - \exp\Big(-\frac{t}{\tau_{\text{fall}}}\Big)\right) \quad , \tag{3.33}$$

where A denotes a constant factor and $\Theta(t)$ the Heaviside step function.
By choosing $\tau_{\text{rise}} = \tau_{\text{fall}} := \tau_{\text{syn}}$ one can simplify this model even more which leads with l'Hôpital's rule to

$$\epsilon(t) = A\Theta(t)\ \frac{1}{\tau_{\text{syn}}^2}\ t \exp\left(-\frac{t}{\tau_{\text{syn}}}\right) \quad . \tag{3.34}$$

This function is called an *alpha function* and the model accordingly *alpha-shaped synapse model*.
Another simplification which is very popular assumes that the diffusion of neurotransmitters is much faster than their removal. Hence one neglects τ_{rise} in (3.33) and sets $\tau_{\text{fall}} = \tau_{\text{syn}}$ which leads to

$$\epsilon(t) = A\Theta(t)\frac{1}{\tau_{\text{syn}}} \exp\left(-\frac{t}{\tau_{\text{syn}}}\right) \quad . \tag{3.35}$$

This model is called *exponential-shaped synapse model*.

3.6.1. Conductance- and Current-based Synapses

We use two different mechanisms to realize synaptic interactions. Considering the LIF model (3.31) we see that the influence due to synaptic connections is modeled by a current I^{syn}. Motivated by this are *current-based* (CUBA) synapse models which represent the synaptic transmission by a current injection into the membrane.
An arguably more biologically realistic approach is motivated by the observation that

in biological synapses incoming spikes cause particular channels in the membrane to open, thereby increasing the membrane conductance towards the corresponding reversal potential. This leads to so-called *conductance-based* (COBA) synapse models.
For the COBA synapses we need to differentiate between excitatory g_e^{syn} and inhibitory g_i^{syn} conductances which causes changes to different reversal potentials E_e^{rev} and E_i^{rev}, respectively. Consequently the synaptic current from (3.31) has the shape

$$I^{\text{syn}} = g_e^{\text{syn}}(E_e^{\text{rev}} - u) + g_i^{\text{syn}}(E_i^{\text{rev}} - u) \quad , \tag{3.36}$$

where g^{syn} replaces the general synaptic interaction f^{syn} which we have introduced in (3.32). The membrane potential ODE (3.31) then becomes

$$C_m \frac{\mathrm{d}u}{\mathrm{d}t} = g_l(E_l - u) + g_e^{\text{syn}}(E_e^{\text{rev}} - u) + g_i^{\text{syn}}(E_i^{\text{rev}} - u) + I^{\text{ext}} \quad . \tag{3.37}$$

We can use the same interaction kernels as discussed before ((3.34), (3.35)) to obtain COBA alpha or COBA exponential-shaped synapses.
We note that in (3.37) the summation of post-synaptic potentials (PSPs) is not linear. They instead depend on the current membrane potential. Furthermore the coupling between the membrane potential u and its time-derivative $\frac{\mathrm{d}u}{\mathrm{d}t}$ is not constant but depends on the time-dependent g_e^{syn} and g_i^{syn}. This makes an analytic treatment of the temporal evolution of the membrane potential much more difficult.

For the CUBA synapse models the general synaptic interaction f^{syn} from (3.32) is represented by the synaptic current I^{syn} itself. The dynamic of the membrane potential stays therefore the same as in (3.31), with the synaptic current

$$I^{\text{syn}}(t) = \sum_{\text{syn}\,k} \sum_{\text{spk}\,s} w_k \Theta(t - t_s)(t - t_s) \exp\left(-\frac{t - t_s}{\tau_{\text{syn}}}\right) \quad , \tag{3.38}$$

in case of alpha-shaped synapses, and

$$I^{\text{syn}}(t) = \sum_{\text{syn}\,k} \sum_{\text{spk}\,s} w_k \Theta(t - t_s) \exp\left(-\frac{t - t_s}{\tau_{\text{syn}}}\right) \tag{3.39}$$

in case of exponential-shaped synapses. One should note that in comparison to the COBA models the synaptic current has no additional dependence on the membrane potential u. Hence the property is preserved, that PSPs are summed up linearly and do not interact with each other. The CUBA models are in summary much easier to handle theoretically but less truthful in representing the behavior of biological synapses. A more detailed description of the synapse models with more information about the biological background can be found in *Petrovici* (2015).

3.7. Tsodyks-Markram Model

The Tsodyks-Markram synapse model (TSO) is a phenomenological model of short-term synaptic plasticity. The model introduces limited synaptic resources which are consumed

when a spike occurs and need to be rebuild afterwards. They could for example represent the total number of vesicles within a synapse. There are two versions of the TSO model. The first was introduced in *Tsodyks and Markram* (1997) and uses three variables to model the evolution of resources. Within this thesis we will only use the newer model described in *Fuhrmann et al.* (2002) and *Maass and Markram* (2002). It is a simplification of the first version and uses just two variables.
In this model, the dynamics of the fraction of synaptic resources $R \in [0, 1]$ is given by

$$\frac{\mathrm{d}R}{\mathrm{d}t} = \frac{(1-R)}{\tau_{\mathrm{rec}}} - U_{SE}\, R\, \delta(t - t_{sp}) \quad , \tag{3.40}$$

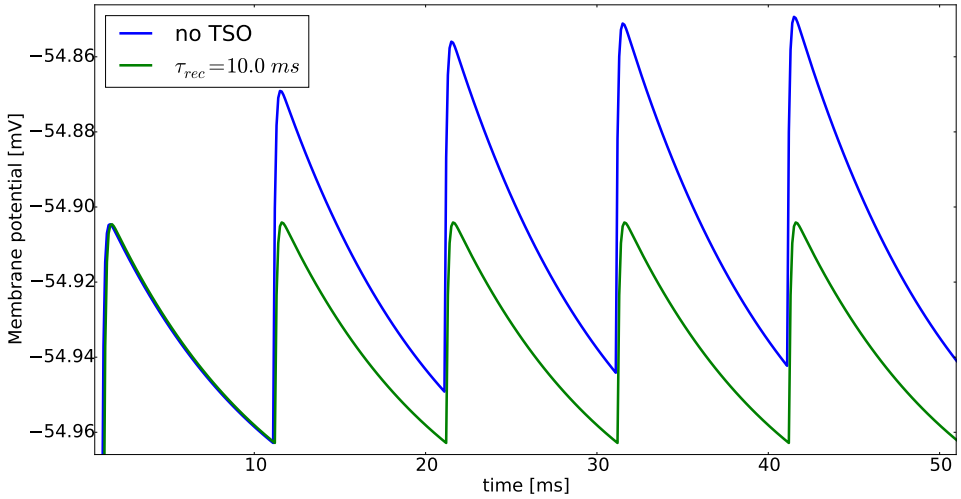
where t_{sp} denotes the time of the occurring pre-synaptic spike, $U_{SE} \in [0, 1]$ stands for the fraction of resources which are utilized for the spike and τ_{rec} is the time constant which determines the recovery of the resources. The maximum amplitude of the post-synaptic response (PSR) will be

$$PSR \propto W \cdot U_{SE} \cdot R \quad , \tag{3.41}$$

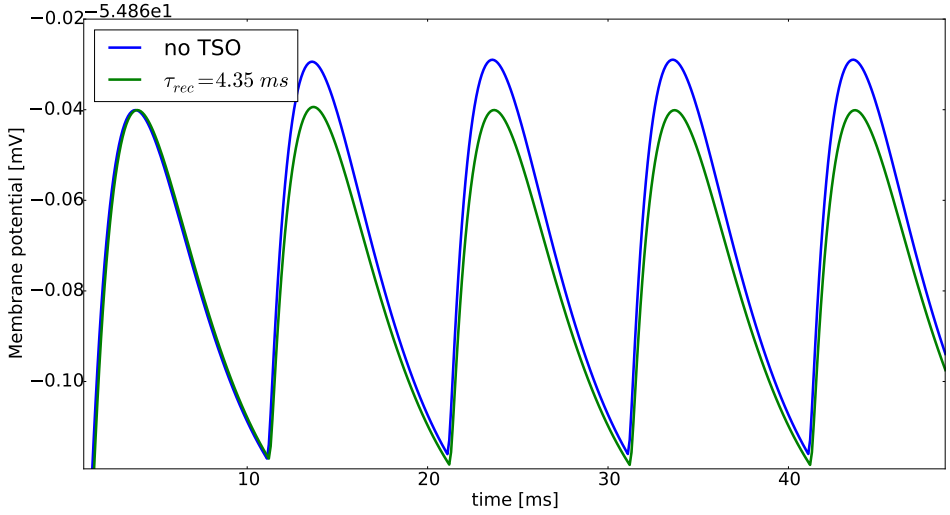
where W stands for the synapse weight.
This mechanism models depressing synapses only. It can also be extended with a mechanism for facilitating synapses. Here one considers U_{SE} as a dynamic variable as well, which evolves like

$$\frac{\mathrm{d}U_{SE}}{\mathrm{d}t} = \frac{U_0 - U_{SE}}{\tau_{\mathrm{facil}}} + U_0\,(1 - U_{SE})\,\delta(t - t_{sp}) \quad , \tag{3.42}$$

where U_0 is the resting state of U_{SE} and τ_{facil} represents the relaxation time constant of the facilitation. This dynamic will make U_{SE} increase at each pre-synaptic spike and decay to its resting state in the absence of spikes.
TSO will play an important role later in this thesis. Initially we applied it to achieve renewing post-synaptic potentials, which are necessary for LIF sampling (Section 3.8). Figure 3.5 shows that this works well using just the synaptic depression with τ_{rec} for both the alpha and exponential-shaped synapse model. In Chapter 5 we will apply even higher recovery time constants to reduce the strength of synaptic connections when a neuron produces many spikes in a short time.



(a) Exponential-shaped synapse model



(b) Alpha-shaped synapse model

Figure 3.5.: Simulation of the membrane potential of a neuron which receives a spike every refractory period with refractory time $\tau_{\text{ref}} = 10\,\text{ms}$. Figure 3.5a shows the result for the exponential-shaped synapse model and Fig. 3.5b for the alpha-shaped one. For the blue curves no TSO was applied. One can see how this yields to a build-up of the PSPs in the beginning. This is due to the fact that the PSP amplitude does not decay to zero after the refractory period, which is demonstrated in Fig. 3.7. The green curves show the simulation with a recovery time τ_{rec} such that this build-up effect is exactly balanced, which leads to a maintenance of the PSP heights. For the exponential-shaped model $\tau_{\text{rec}} = 10\,\text{ms}$ is necessary while for the alpha-shaped model $\tau_{\text{rec}} = 4.35\,\text{ms}$ is sufficient because the build-up in this model is weaker.

3.8. LIF Sampling

Building upon the neural sampling theory (Section 3.4), in *Petrovici et al.* (2013), an approach was developed to perform sampling with spiking LIF neurons. The LIF model (Section 3.5) is a deterministic neuron model. The sampling theory, though, depends on the stochastic nature of their units. Hence, we need to add stochasticity to the system. To realize this, we add connections to spiking Poisson sources to each neuron.[1] Poisson sources are additional units which spike randomly such that the number of spikes follows a Poisson distribution:

$$p(n_2, t_2|n_1, t_1) = \frac{(\nu\Delta t)^{\Delta n}}{\Delta n!}\mathrm{e}^{-\nu\Delta t} \quad , \tag{3.43}$$

where ν is the firing rate of the Poisson source and $\Delta n = n_2 - n_1$ the number of spikes it generates in the time interval $\Delta t = t_2 - t_1$. One can reformulate the equation for the membrane potential u_k of a single LIF neuron (3.31) with COBA synapses (Section 3.6) to the following form

$$\tau_{\mathrm{eff}}(t)\frac{\mathrm{d}u_k}{\mathrm{d}t} = u_k^{\mathrm{eff}}(t) - u_k \tag{3.44}$$

$$u_k^{\mathrm{eff}}(t) = \frac{g_L E_L + \sum_i g_i^{\mathrm{syn}} E_i^{\mathrm{rev}} + I^{\mathrm{ext}}}{g_{\mathrm{tot}}(t)} \tag{3.45}$$

$$\tau_{\mathrm{eff}}(t) = \frac{C_m}{g_{\mathrm{tot}}(t)} \tag{3.46}$$

$$g_{\mathrm{tot}}(t) = g_L + \sum_i g_i^{\mathrm{syn}}(t) \quad . \tag{3.47}$$

The sum over the index i is in this case a sum over the Poisson sources. If one chooses sources with a high firing rate ν the neuron enters the so-called *high conductance state* (HCS). It is characterized by a high g_{tot} due to the large synaptic input from the Poisson sources. This leads to a low τ_{eff} and therefore to a quickly reacting membrane such that $u_k(t) \approx u_k^{\mathrm{eff}}(t)$. Hence the real membrane potential of the neuron follows the effective or also called *free membrane potential* u_k^{eff}.
For CUBA synapses, the conductance is not changed due to the Poisson input. Here we have $g_{\mathrm{tot}} = g_l$ and therefore $\tau_{\mathrm{eff}} = \tau_m := \frac{C_m}{g_l}$. To obtain a similar behavior as in the HCS here, we need to choose the parameters such that τ_m becomes very small. Apart from that the behavior is similar.
In *Petrovici et al.* (2013) it has been shown that the free membrane potential follows an Ornstein-Uhlenbeck (OU) process in the HCS. The OU process is a first-order Markov process which can also be used to model the position of a particle in a harmonic potential with diffusion. It contains a deterministic *drift* part, e.g. due to the harmonic potential, and a stochastic *diffusion* part which can be caused e.g. by Brownian motion. Its

[1] In practice, we balance the background noise by connecting each neuron to an excitatory and an inhibitory Poisson source.

stochastic differential equation for the membrane potential is given by

$$\mathrm{d}u(t) = \theta\,(\mu - u(t))\mathrm{d}t + \sigma\,\mathrm{d}W(t) \quad . \tag{3.48}$$

The first part describes the drift of the potential to its mean value μ and the second part the diffusion with an increment $\mathrm{d}W(t)$ of the *Wiener process*. The constants for the COBA case can be calculated to be

$$\theta = \frac{1}{\tau_{\mathrm{syn}}} \tag{3.49}$$

$$\mu = \frac{I^{ext} + g_l E_l + \sum_i \nu_i w_i E_i^{\mathrm{rev}} \tau_{\mathrm{syn}}}{\langle g_{\mathrm{tot}} \rangle} \tag{3.50}$$

$$\sigma^2 = \frac{\sum_i \nu_i \Big[w_i (E_i^{\mathrm{rev}} - \mu) \Big]^2 \tau_{\mathrm{syn}}}{\langle g_{\mathrm{tot}} \rangle^2} \quad . \tag{3.51}$$

The advantage of showing that the free membrane potential approximately evolves according to the OU process is that it is theoretically well understood. It is therefore easy to find analytic expressions for the probability distribution of the membrane potential and the mean first passage time $\langle T(b,a) \rangle$, which is the mean time a process reaches a value "b" starting at "a". This gives us the means to derive the activation function of the neuron, which is the probability to find the neuron in the active state at a random time point.

In the HCS the dynamic of the neuron can be divided into two modes which are illustrated in Fig. 3.6. One is the bursting mode, where the free membrane potential stays above the threshold over a duration of more than one spike. Here due two the fast dynamics of the HCS the real membrane potential will nearly immediately follow it, which makes the neuron spike approximately every refractory period.

The second mode is the freely evolving mode, where the free membrane potential stays below the threshold and the real membrane potential follows it. Figure 3.6 B shows also the behavior of the free membrane potential. Given these modes, one can express the activation function as

$$p(z_k = 1) = \frac{\sum_n P_n\, n\, \tau_{\mathrm{ref}}}{\sum_n P_n \left(n\, \tau_{\mathrm{ref}} + \sum_{k=1}^{n-1} \overline{\tau_k^b} + T_n \right)} \quad , \tag{3.52}$$

where P_n denotes the probability that a spike burst of length n occurs, T_n the time in which the neuron evolves freely after a burst of length n, τ_{ref} the refractory period and $\overline{\tau_k^b}$ the average time the membrane needs to reach the threshold starting from the reset potential after the k^{th} refractory period within a burst. In the HCS $\overline{\tau_k^b}$ is typically very small and will therefore be neglected in the following. The supplementary material of *Petrovici et al.* (2013) describes its influence in more detail. We can calculate the P_n's with the knowledge about the probability distribution as illustrated in Fig. 3.6 B. Here, the red distributions represent the Gaussian distribution of the free membrane potential in the OU process. At the end of each refractory period we can calculate the

probability to fire again by integrating over the part of the distribution which lies above the firing threshold. The average duration of the freely evolving membrane potential T_n can be calculated with the mean first passage time of the OU process. This enables us to calculate an analytic expression for the activation function. A detailed derivation can be found in *Petrovici* (2015) and in the supplementary material of *Petrovici et al.* (2013).

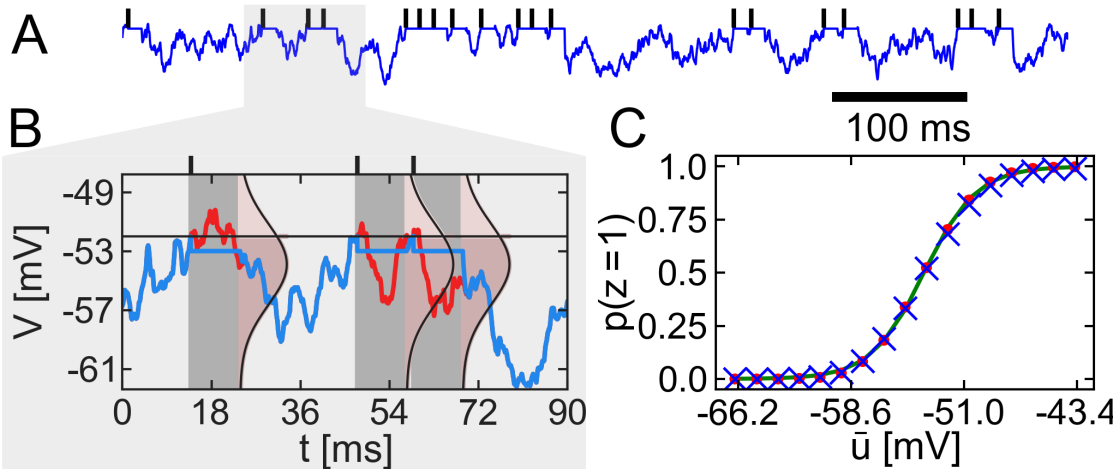


Figure 3.6.: **A**: Evolution of the real membrane potential of a neuron (blue line). The spike times are marked with black lines. One can also see the refractory periods after the spikes where the membrane potential is set to a fixed value.
B: An enlarged part of A. The blue line shows again the real membrane potential and the red line the free one which evolves according to an OU process. The gray regions denote the refractory periods. The red Gaussian distributions represent the probability distribution of the free membrane potential. The lighter part above the threshold indicates the probability to spike again after the end of a refractory period.
C: The activation function of an LIF neuron over its mean membrane potential. The red dots show the theoretical prediction based on (3.52). The blue crosses show the results of a simulation and the green curve is a logistic function which was fitted to the theoretical prediction. Image taken from *Petrovici et al.* (2013).

The shape of the activation function is displayed in Fig. 3.6 C. In the HCS, the activation function has a symmetric sigmoidal shape, as it is required from the neural sampling theory (3.30). To obtain the necessary logistic activation function we still need to shift and rescale the sigmoid such that the activation function becomes

$$p(z_k = 1) = \sigma\left(\frac{\bar{u}_k - \bar{u}_k^0}{\alpha}\right) \quad , \tag{3.53}$$

where the parameters $\bar{u}_k^0$ and α can be obtained by fitting (3.53) to the simulated activation function. We need to consider this as well to convert the neuron bias between the

LIF and abstract regime

$$b_k = (\bar{u}_k^b - \bar{u}_k^0)/\alpha \quad . \tag{3.54}$$

To translate the LIF weight w_{kj} of the neuron k to some other neuron j to the corresponding weight in the abstract regime W_{kj}, we need to take also the synapse model into account. In the abstract regime (3.29) the post-synaptic potential (PSP) of a spike is just a rectangular box. In the LIF regime it has a different shape which is determined by the synapse model. The shapes of the PSPs for the alpha and exponential-shaped synapse are shown in Fig. 3.7. One can see the clear difference which will lead to a systematic error in the behavior for LIF synapses compared to the abstract regime. To limit this error, one chooses the weight translation such that the value of the PSP is at least on average equal to the box value. Hence, the integral of the PSP shape until the refractory period should be equal to the area of the box. The horizontal line in Fig. 3.7 marking the box-shaped PSP is chosen to fulfill this. With this condition, one obtains for the weight translation between the regimes in case of exponential-shaped synapses

$$W_{kj} = \frac{1}{\alpha C_m} \frac{w_{kj}\left(E_{kj}^{\text{rev}} - \mu\right)}{1 - \frac{\tau_{\text{syn}}}{\tau_{\text{eff}}}} \left[\tau_{\text{syn}}\left(\mathrm{e}^{-1} - 1\right) - \tau_{\text{eff}}\left(\mathrm{e}^{-\frac{\tau_{\text{syn}}}{\tau_{\text{eff}}}} - 1\right)\right] \quad . \tag{3.55}$$

See *Petrovici et al.* (2013) for a detailed calculation.
With the correct activation function and the rules for the weight translation between the LIF and abstract regime it is possible to perform the same sampling tasks as with neural sampling. This makes especially sampling in Boltzmann machines (Section 3.2) with networks of LIF neurons possible. We therefore refer to these networks as *LIF-based Boltzmann machines*. LIF sampling was already used before for tasks like Bayesian inference (*Probst*, 2014; *Probst et al.*, 2015), classification (*Leng*, 2014; *Roth*, 2014) and pattern completion (*Roth*, 2014).

3.9. Training Restricted Boltzmann Machines

We introduced the Boltzmann machine (BM) and in particular the restricted Boltzmann machine (RBM) in Section 3.2. The aim of training/learning a BM is to make it represent a good model of the distribution underlying the training data. The general idea is therefore to maximize the probability of the BM to reproduce the training data. This is the same for BMs and RBMs. In this thesis only training of RBMs will be used. Therefore we will describe the training approach in the following mainly for RBMs. A more detailed introduction with practical examples to this can be found in *Fischer and Igel* (2014).
For RBMs only the visible layer represents the input data. This means that during training we show the RBM the training data by clamping the values of the visible layer to the training data. The visible layer therefore needs to have the same state space as the input data. By changing the parameters of the RBM we aim to maximize the marginal

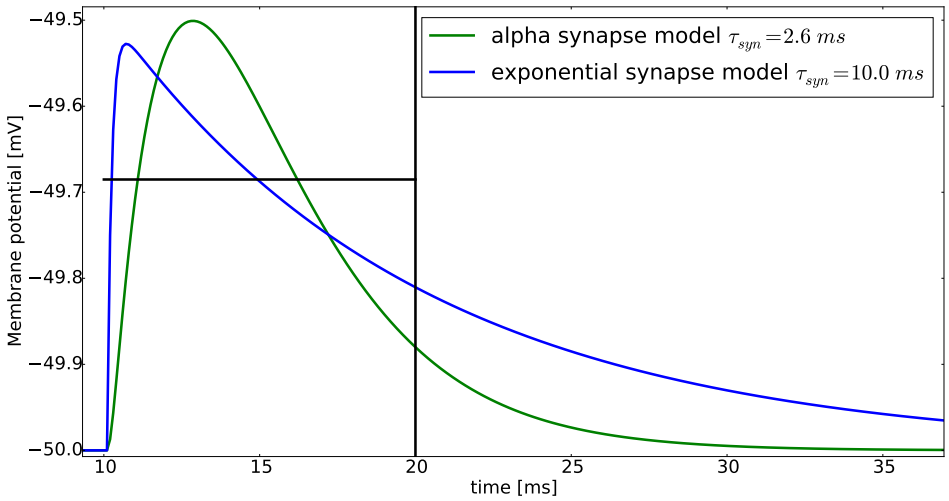


Figure 3.7.: Simulated shape of the post-synaptic potential for an alpha and exponential-shaped synapse. The vertical black line denotes the end of the refractory period of the neurons and the horizontal line shows the corresponding PSP for the abstract regime. The height of the line is chosen such that the integral under the box is equal to the integral of the synaptic PSPs. The area over the box for the alpha-shaped model is approximately 1.57 times larger than the corresponding area for the exponential-shaped model.

distribution of the visible layer

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \quad . \tag{3.56}$$

Since the logarithm is monotonic, $\max(p) \mathrel{\widehat{=}} \max(\log p)$, it is sufficient to maximize the log-likelihood

$$\ln p(\mathbf{v}) = \ln \left(\sum_{\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \right) - \ln \left(\sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \right) \quad . \tag{3.57}$$

Taking the derivative of the log-likelihood with respect to the weights of the RBM yields

$$\frac{\partial \ln p(\mathbf{v})}{\partial W_{ij}} = \frac{\sum_{\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \left(-\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial W_{ij}} \right)}{\sum_{\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})}} - \frac{\sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})} \left(-\frac{\partial E(\mathbf{v},\mathbf{h})}{\partial W_{ij}} \right)}{\sum_{\mathbf{v},\mathbf{h}} \mathrm{e}^{-E(\mathbf{v},\mathbf{h})}} \tag{3.58}$$

$$= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}) v_i h_j - \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}) v_i h_j \quad , \tag{3.59}$$

where in the second term in the last step we identified the joint distribution of the RBM and in the first term we used

$$p(\mathbf{h}|\mathbf{v}) = \frac{p(\mathbf{v},\mathbf{h})}{p(\mathbf{v})} = \frac{\frac{1}{Z}\mathrm{e}^{-E(\mathbf{v},\mathbf{h})}}{\frac{1}{Z}\sum_{\mathbf{h}}\mathrm{e}^{-E(\mathbf{v},\mathbf{h})}} = \frac{\mathrm{e}^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{h}}\mathrm{e}^{-E(\mathbf{v},\mathbf{h})}} \quad . \tag{3.60}$$

One often regards the mean of the derivative over a training set $S = \{\mathbf{v}'_1, \ldots, \mathbf{v}'_K\}$ where each $\mathbf{v}'_i$ denotes a input data point to which the visible layer is clamped to. This leads to

$$\frac{1}{K}\sum_{\mathbf{v}'}\frac{\partial \ln p(\mathbf{v}')}{\partial w_{ij}} = \frac{1}{K}\sum_{\mathbf{v}'}\left(\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v}')v'_i h_j - \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h})v_i h_j\right) \tag{3.61}$$

$$= \langle v'_i h_j\rangle_{p(\mathbf{h}|\mathbf{v}')q(\mathbf{v}')} - \langle v_i h_j\rangle_{p(\mathbf{v},\mathbf{h})} \quad , \tag{3.62}$$

where q denotes the training data distribution. We note that, as we sum over all possible states in the second term, it is independent from the input data. This leads us to the well-known maximum likelihood learning rule

$$\left\langle \frac{\partial \ln p(\mathbf{v})}{\partial w_{ij}} \right\rangle = \langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{model} \quad . \tag{3.63}$$

For an efficient calculation of the derivative of the log-likelihood, we can simplify (3.59) by making use of the restricted topology of the RBM. As there are no connections within the hidden layer, the marginal hidden distribution factorizes. The first term in (3.59) therefore becomes

$$\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})v_i h_j = \sum_{h_j\in\{0,1\}}\sum_{\mathbf{h}_{\setminus j}} p(h_j|\mathbf{v})p(\mathbf{h}_{\setminus j}|\mathbf{v})v_i h_j \tag{3.64}$$

$$= \sum_{h_j\in\{0,1\}} p(h_j|\mathbf{v})v_i h_j \underbrace{\sum_{\mathbf{h}_{\setminus j}} p(\mathbf{h}_{\setminus j}|\mathbf{v})}_{=1} \tag{3.65}$$

$$= p(h_j = 1|\mathbf{v})v_i \quad . \tag{3.66}$$

With (3.66) one obtains for the derivative of the log-likelihood (3.59)

$$\frac{\partial \ln p(\mathbf{v})}{\partial W_{ij}} = \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})v_i h_j - \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h})v_i h_j \tag{3.67}$$

$$= \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})v_i h_j - \sum_{\mathbf{v}} p(\mathbf{v})\sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v})v_i h_j \tag{3.68}$$

$$= p(h_j = 1|\mathbf{v})v_i - \sum_{\mathbf{v}} p(\mathbf{v})p(h_j = 1|\mathbf{v})v_i \quad . \tag{3.69}$$

We use this derivative to apply a gradient ascent rule to optimize the weights for the log-likelihood

$$W_{ij}^{(t+1)} = W_{ij}^{(t)} + \eta\frac{\partial \ln p(\mathbf{v})}{\partial W_{ij}^{(t)}} \quad , \tag{3.70}$$

where $\eta \in \mathbb{R}_{>0}$ is called *learning rate*.
To train the biases we use the same procedure. A similar calculation yields for the derivative of the log-likelihood with respect to the bias of the visible units

$$\frac{\partial \ln p(\mathbf{v})}{\partial a_i} = v_i - \sum_{\mathbf{v}} p(\mathbf{v}) v_i \tag{3.71}$$

and for the bias of the hidden units

$$\frac{\partial \ln p(\mathbf{v})}{\partial b_j} = p(h_j = 1|\mathbf{v}) - \sum_{\mathbf{v}} p(\mathbf{v}) p(h_j = 1|\mathbf{v}) \quad . \tag{3.72}$$

3.9.1. Contrastive Divergence

Despite the simplifications due to the RBM topology the computation of the learning rules (3.69), (3.71) and (3.72) is in most cases computationally intractable because we have to sum over all possible visible states. For example in the MNIST data set (*LeCun and Cortes*, 1998) there are 2^{784} possible visible states. Hence we need to approximate this term.
One simple approach for this is called *contrastive divergence* (CD) (*Hinton*, 2002). CD uses Gibbs sampling (Section 3.3.2) to get a rough approximation of the model term in the learning rules. For k-step CD or short CD-k we exchange in the second term in (3.69) the expectation with respect to $p(\mathbf{v})$ with the k^{th} Gibbs sample of $\mathbf{v}$. Hence the derivative of the log-likelihood $\frac{\partial \ln p(\mathbf{v})}{\partial W_{ij}}$ is replaced with the expression

$$\mathrm{CD}_k(\mathbf{v}^{(0)}, W_{ij}) = \; p(h_j = 1|\mathbf{v}^{(0)}) v_i^{(0)} - p(h_j = 1|\mathbf{v}^{(k)}) v_i^{(k)} \quad , \tag{3.73}$$

where $\mathbf{v}^{(0)}$ is the original visible layer clamped to the input data and $\mathbf{v}^{(k)}$ the visible layer after k Gibbs sampling steps. The log-likelihoods with respect to the biases (3.71), (3.72) are correspondingly replaced with

$$\mathrm{CD}_k(\mathbf{v}^{(0)}, a_i) = \; v_i^{(0)} - v_i^{(k)} \tag{3.74}$$

for the visible units and

$$\mathrm{CD}_k(\mathbf{v}^{(0)}, b_j) = \; p(h_j = 1|\mathbf{v}^{(0)}) - p(h_j = 1|\mathbf{v}^{(k)}) \tag{3.75}$$

for the hidden units.
Despite this rough approximation, training works often well even when setting $k = 1$.

3.9.2. Persistent Contrastive Divergence

Persistent contrastive divergence (PCD) (*Tieleman*, 2008) is an approach to enhance normal CD. To approximate the second term in (3.69) PCD uses a persistent chain. This means that in contrast to CD-k we do not reinitialize the Markov chain of the Gibbs sampler every time we consider a new data point $\mathbf{v}^{(0)}$. Instead we just let the Markov

chain run k further steps for every data point. The Markov chain is only initialized with the first training data point. This replaces, except for the beginning of the training, the sample $\mathbf{v}^{(k)}$ in (3.73) with a sample that is independent of $\mathbf{v}^{(0)}$.
The idea behind this is that if the learning rate is sufficiently small, which leads to just small changes of the model for each parameter update, one can assume that the Markov chain stays close to the stationary distribution. Hence, it produces a better approximation of the current model distribution.
This is the point where mixing (Section 3.3.3) becomes important during training, because for small learning rates it is possible that the Markov chain gets stuck in a subspace of the distribution and therefore produces just bad estimates of the stationary distribution. Indeed, in *Welling et al.* (2003); *Tieleman and Hinton* (2009); *Salakhutdinov* (2009) it has been shown that bad mixing leads to unstable learning dynamics and poor parameter estimates.
To train the MNIST examples in (Chapter 4, Chapter 5) we always used PCD.

3.10. Data Visualization

In Section 4.3 we need to evaluate the high-dimensional sampling results generated from Boltzmann machines trained on the MNIST data set. One approach to achieve this is to visualize the high-dimensional data samples by assigning each sample a location in a two or three-dimensional map. The main problem thereby is to reduce the dimensionality but at the same time preserve the significant structure of the high-dimensional data. In the following, we will present the visualization techniques which will be used within this thesis.

3.10.1. Star Plot

To visualize results for simulations with RBMs trained on just 3 MNIST digits, we use so-called *star plots* as described in *Petrovici et al.* (2013).
We consider a set of N samples $\mathcal{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N\}$ of the visible layer of an RBM. The vectors for the three training digits are denoted by $\mathbf{B}^0, \mathbf{B}^3, \mathbf{B}^4$. As in the experiments later (Section 4.3.1), we assume that we train on the digits 0, 3, 4.
In a first step, we project each sample $\mathbf{z}_i \in \mathcal{Z}$ linearly on three dimensions using the training digit vectors as a basis

$$\mathbf{z}_i^{034} = \begin{pmatrix} \mathbf{B}^0 \cdot \mathbf{z}_i \\ \mathbf{B}^3 \cdot \mathbf{z}_i \\ \mathbf{B}^4 \cdot \mathbf{z}_i \end{pmatrix} \quad . \tag{3.76}$$

Afterwards, we project this three-dimensional vector into two dimensions by assigning each of the three digits an angle $(\phi^0, \phi^3, \phi^4) = (0, \frac{2\pi}{3}, \frac{4\pi}{3})$ on the unit circle

$$\mathbf{z}_i^{\text{proj}} = \begin{pmatrix} \sin(\phi^0) & \sin(\phi^3) & \sin(\phi^4) \\ \cos(\phi^0) & \cos(\phi^3) & \cos(\phi^4) \end{pmatrix} \mathbf{z}_i^{034} \quad . \tag{3.77}$$

This method is simple and yields good results for the case of an RBM trained on 3 digits as can bee seen in Section 4.3.1. For larger training sets, it becomes unfeasible. If, for example, we train an RBM on 3 digits with 5 images per digit, we have to use the mean of the 5 images to determine the basis vector for the digit. This leads to a larger variation of the projected states around the direction representing the classes and therefore often to overlaps between the classes. As this happens already for such small examples, we conclude that more sophisticated approaches are necessary to visualize samples from larger systems. Such approaches will be discussed in the following sections.

3.10.2. Principal Component Analysis (PCA)

Principal Component Analysis PCA (*Hotelling*, 1933), is widely used for applications such as data visualization, dimensionality reduction, feature extraction and lossy data compression. It is described in many textbooks such as *Bishop and Nasrabadi* (2006); *Jolliffe* (2002). Hence we will give just a brief description here, which focuses on the intuition behind this method.
PCA finds a linear projection from a high-dimensional data space onto a lower-dimensional subspace which captures as much variance as possible. One assumes that the high-dimensional data is approximately distributed according to a multivariate Gaussian. Hence the data points will roughly lie within an N-dimensional ellipsoid. The eigenvectors of the covariance matrix of the data set are in this case oriented along the main axes of the ellipsoid and the square roots of the corresponding eigenvalues are proportional to the variances in these directions.
An example for this in two dimensions is shown in Fig. 3.8. To reduce the data to $K < N$ dimensions, we sort the eigenvectors corresponding to the size of their eigenvalues and take the K first eigenvectors (principal components). These define the directions in the N-dimensional space with the largest variance.
Finally we apply a linear mapping of the data onto these eigenvectors. This yields the map of our data set in K-dimensions with the largest possible variance. In case of Fig. 3.8, this means that if we intend to reduce the dimension of the data to one, we apply a scalar product between the data points and eigenvector of the major axis (red arrow). Hence one can imagine PCA as fitting an N-dimensional ellipsoid to the data and subsequently omitting the axes with the lowest variance.

We will use PCA as a reference method to the more complicated t-SNE approach (Section 3.10.4), because it is a standard method and very easy to implement and to handle, as it involves no parameter tuning. If we obtained satisfying results with PCA, there would be no need to apply a much more complicated method like t-SNE.

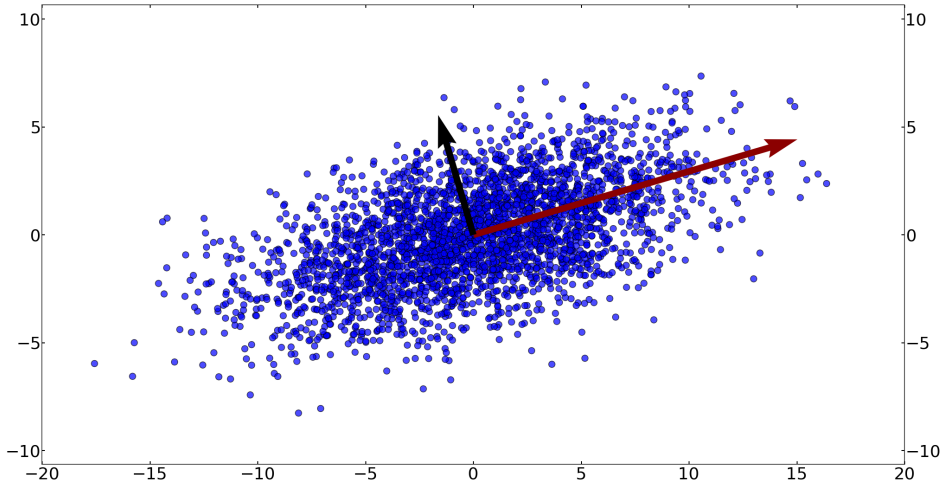


Figure 3.8.: Data distributed according to a multivariate Gaussian distribution. The arrows show the directions of the eigenvectors of the covariance matrix. The length of the arrows are proportional to the square root of the corresponding eigenvalues.
The x-coordinates of the data points were drawn from a Gaussian distribution. The y-coordinates consist of a fraction of the x-coordinates plus random Gaussian noise.

3.10.3. Stochastic Neighbor Embedding (SNE)

Stochastic neighbor embedding (SNE) was introduced in *Hinton and Roweis* (2002). As in PCA, it tries to find a low-dimensional map for a high-dimensional data set. The new notion of this method is embedding data points within a set based on probable neighbors. We denote the high-dimensional data set as $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and the two (or three) dimensional map as $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$. The high-dimensional set $\mathcal{X}$ is fixed to the input data, which are in our case samples from a Boltzmann distribution. The low-dimensional coordinates are free to move.
In SNE, one begins with converting the Euclidean distances between high-dimensional data points into conditional probabilities that represent their similarities. We assume the similarity of data point $\mathbf{x}_j$ to data point $\mathbf{x}_i$ to be represented by a Gaussian conditional probability

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} \quad , \tag{3.78}$$

where σ_i is the variance of the Gaussian that is centered on data point $\mathbf{x}_i$. $p_{j|i}$ can also be read as the probability that the data point $\mathbf{x}_i$ would pick $\mathbf{x}_j$ as its neighbor.
In the low-dimensional map, we model the similarity of point $\mathbf{y}_j$ and $\mathbf{y}_i$ as well as a

Gaussian but this time with a fixed variance set to $\frac{1}{2}$

$$q_{j|i} = \frac{\exp(-\|\mathbf{y}_i - \mathbf{y}_j\|^2)}{\sum_{k \neq i} \exp(-\|\mathbf{y}_i - \mathbf{y}_k\|^2)} \quad . \tag{3.79}$$

If $\mathbf{y}_i$ and $\mathbf{y}_j$ correctly model the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$, the conditional probabilities $p_{j|i}$ and $q_{j|i}$ will be equal.
Thus, the aim of SNE is to find a low-dimensional data representation that minimizes the mismatch between all possible combinations of $p_{j|i}$ and $q_{j|i}$. We apply the Kullback-Leibler divergence (D_{KL})(Section 3.1) as a measure for this mismatch and define the cost function C as

$$C = \sum_i D_{KL}(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \quad , \tag{3.80}$$

where P_i represents the conditional probability distribution over the possible $p_{j|i}$'s given one $\mathbf{x}_i$ and Q_i represents the corresponding distribution for the $q_{j|i}$'s.
The unsymmetric nature of the D_{KL} leads to a different weighting of possible errors. Using widely separated map points (small $q_{j|i}$) for close data points (large $p_{j|i}$) leads to a large cost. In contrast to this, using close map points (large $q_{j|i}$) to represent largely separated data points (small $p_{j|i}$), leads to a relatively small cost, which is caused by wasting some of the probability mass of q. Therefore, for widely separated data points (small $p_{j|i}$) it is relatively unimportant to choose small $q_{j|i}$ as well. Minimizing the cost function consequently focuses on a good representation of the local data structure.

For modeling the high-dimensional data point similarities (3.78) it is important to select reasonable σ_i's for the Gaussians. In dense regions one should choose a small σ_i because as there are many data points close together we want the probability that a data point would choose another as its neighbor to decay rapidly with their difference. In sparse regions, on the contrary, one should choose a large σ_i. To achieve these conditions, it is desirable to fix something like the effective number of neighbors for each data point. To find such a quantity, one can consults information theory. Each variance σ_i determines the distribution P_i over the single $p_{j|i}$'s for a data point $\mathbf{x}_i$.
The *Shannon entropy* of P_i is defined as

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i} \quad . \tag{3.81}$$

Based on it, we can define the so-called *perplexity*

$$Perp(P_i) = 2^{H(P_i)} \quad . \tag{3.82}$$

The perplexity of a probability distribution is a measure for how well it predicts a sample. For example the perplexity of an unbiased die with k possible outcomes is k according to (3.82). Hence, if a data point has k data points with the same distance (k neighbors), we would have the same situation. Thus the perplexity would be k. Therefore one can interpret the perplexity as a smooth measure of the effective number of neighbors.

Typical values for the perplexity are chosen to be 5 - 50. As the perplexity is like the entropy monotonically increasing with σ_i, it is possible to find the σ_i's for a chosen perplexity via *binary search*. The perplexities thereby are interpreted as the target search values and the variances as their corresponding keys.

It can be shown that the gradient of (3.80) with respect to the coordinates of a map point i is

$$\frac{\partial C}{\partial \mathbf{y}_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}) (\mathbf{y}_i - \mathbf{y}_j) \quad . \tag{3.83}$$

Physically this can be interpreted as the resultant force of a set of springs connecting $\mathbf{y}_i$ to all other map points $\mathbf{y}_j$. The stiffness of the springs is determined by the mismatch between the pairwise similarities of data and map points.
There are many possibilities to minimize the cost function. In *Van der Maaten and Hinton* (2008) the minimization is performed by gradient descent.
The gradient descent is initialized by sampling map points randomly from an isotropic Gaussian with small variance that is centered around the origin. To avoid poor local minima a momentum to the gradient update rule is added

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\partial C}{\partial \mathcal{Y}^{(t-1)}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right) \quad , \tag{3.84}$$

where $\mathcal{Y}$ denotes the solution for the full set of map points at iteration t, $\alpha(t)$ the momentum and η the learning rate .
In *Van der Maaten and Hinton* (2008) it was observed that this gradient is difficult to optimize which made it necessary to additionally add Gaussian noise to the map points after each iteration in the early stages of the optimization. The variance of the noise is gradually reduced, which makes it corresponding to a *simulated annealing* that helps to escape from poor local minima.

3.10.4. t-Distributed Stochastic Neighbor Embedding (t-SNE)

In *Van der Maaten and Hinton* (2008) a new technique called *t-Distributed Stochastic Neighbor Embedding* (t-SNE) was presented, which is a variation of SNE. t-SNE addresses two problems of SNE. The cost function, which is difficult to optimize and the so-called crowding problem. To achieve this it uses a symmetrized version of the SNE cost function with simpler gradients and a t-Student distribution to model the similarities between map points. Both will be explained in the following.

Symmetrization of the SNE Cost Function

We exchange the conditional similarities between the high-dimensional data points (3.78) and between the map points (3.79) with symmetric pairwise similarities. For the high-dimensional data points we use

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \quad , \tag{3.85}$$

where n is the number of data points.
And for the map points

$$q_{ij} = \frac{\exp(-\|\mathbf{y}_i - \mathbf{y}_j\|^2)}{\sum_{k \neq l} \exp(-\|\mathbf{y}_k - \mathbf{y}_l\|^2)} \quad . \tag{3.86}$$

Note that the sum in the denominator goes over both k and l.
The resulting cost function is similar to (3.80)

$$C = D_{KL}(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \quad . \tag{3.87}$$

It can be shown that its gradient has the following shape which is also similar to the one of asymmetric SNE

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_j (p_{ij} - q_{ij})(\mathbf{y}_i - \mathbf{y}_j) \quad . \tag{3.88}$$

The main advantage of this gradient is that it is simpler and faster to compute. According to *Van der Maaten and Hinton* (2008) this method produces maps that are as good or sometimes a little better than for the normal asymmetric SNE.

As a side remark, in (3.85) one may wonder why no Gaussian is chosen to model the symmetric similarities between high-dimensional data points similarly to

$$p_{ij} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)}{\sum_{k \neq l} \exp(-\|\mathbf{x}_k - \mathbf{x}_l\|^2/2\sigma^2)} \quad . \tag{3.89}$$

To motivate this, we consider a high-dimensional data point $\mathbf{x}_k$ which is an outlier. This means that it has a high distance to all other points. As a consequence of (3.89), all probabilities p_{kj} of the point to its neighbors become very small. Hence, this point has nearly no influence on the cost function (3.87). As a result, the position of the corresponding map point $\mathbf{y}_k$ will be barely determined leading to a bad mapping for this kind of points.
This can be avoided by choosing (3.85). Because due to the definition of $p_{i|j}$ (3.78) we have $\sum_j p_{ij} = \sum_j \frac{p_{j|i} + p_{i|j}}{2n} > \frac{1}{2n}$ for all data points $\mathbf{x}_i$. Hence all points will have a significant representation in the cost function.

Solving the Crowding Problem with Heavy-Tailed Distributions

A correct model of high-dimensional distances in low-dimensional space is hindered by several problems. One is that in D dimensions it is possible to have $D + 1$ mutually equidistant data points. Hence, it will become impossible to map a high-dimensional data set with several hundred equidistant points correctly to e.g. two dimensions.
Furthermore the volume of a sphere around a data point i scales with r^D. Thus, in high-dimensional space one has far more space to model large distances to this point than in low-dimensional space. Therefore, if we try to represent both, low and large distances,

faithfully and use a reasonable separation for the low distance points, the large distance points need to be placed very far away in the map.
Many algorithms for dimensionality reduction including SNE cannot handle this and suffer therefore from the so-called *crowding problem* where distant data points are not separated very well and the whole map tends to be crowded in the origin.
The approach to solve this issue in t-SNE is to still use Gaussians for high-dimensional probabilities but the *Student's t-distribution* for the low-dimensional ones. The t-distribution for a value $x \in \mathbb{R}$ has the following shape

$$p(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\,\Gamma(\frac{\nu}{2})} \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}} \quad , \tag{3.90}$$

where $\Gamma(x)$ is the Gamma function and $\nu > 0$ is called the *degrees of freedom* of the t-distribution. For $\nu \to \infty$ the t-distribution becomes the Gaussian distribution and for $\nu = 1$ one obtains the Cauchy distribution (Lorentz distribution). Hence, compared to the Gaussian distribution the t-distribution with small ν has much heavier tails while the center of the distribution is similar. Thus, for two data points with a large distance between each other (small p_{ij}) we have to choose a much higher distance to obtain a similar value for q_{ij} in the t-distribution. However, for two close data points (large p_{ij}) there will be no big difference in the distance for q_{ij}.
As a consequence, we exaggerate just larger distances in the map as it is necessary to mitigate the crowding problem.
As for larger ν the t-distribution becomes more like a Gaussian, one can use ν to tune the exaggeration of large distances in the map.
In our case we use a t-distribution with one degree of freedom (Cauchy distribution)

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} \quad . \tag{3.91}$$

Applying the same cost function as in (3.87) leads to the gradient

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_j (p_{ij} - q_{ij})(\mathbf{y}_i - \mathbf{y}_j)(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1} \quad . \tag{3.92}$$

The additional $(1+\|\mathbf{y}_i-\mathbf{y}_j\|^2)^{-1}$-term reduces the "force" between map points with large distances, which makes them less prone to be crowded.
According to *Van der Maaten and Hinton* (2008) this gradient leads to a simpler optimization. As a consequence, in t-SNE one can abandon the thermal annealing which is usually used in SNE.
As an enhancement for t-SNE on the MNIST data set a method called *early exaggeration* is applied. Here one multiplies all p_{ij}'s by a factor $f > 1$ in the beginning. This forces the map points two form tight clusters to achieve large values for the corresponding q_{ij}'s as well. There is consequently more space between the clusters in the beginning which alleviates their movement. This makes it more likely to find a good arrangement for them. For the simulations in this thesis we have applied early exaggeration of $f = 4$ for the first 100 steps.

4. Visualization of Mixing in Generated Data

In this chapter we will investigate the behavior of sampling algorithms in Boltzmann machines (BMs) and restricted Boltzmann machines (RBMs) (Section 3.2). We are especially interested in the ability of the sampling algorithms to move around the state space. This *mixing* property is discussed in Section 3.3.3 and has, in some circumstances, a strong influence on the ability of the sampling algorithms to reproduce their stationary distributions. Finding out how strong this influence is and under which circumstances it becomes important is the aim of this chapter. To be able to judge this we will try out different methods to visualize mixing for the sampling algorithms.
The algorithm in which we are especially interested in is LIF sampling (Section 3.8) because it can, in principle, be mapped on neuromorphic hardware devices as discussed in Section 2.3. We will therefore consider the sampling results for LIF sampling with exponential and alpha-shaped current-based (CUBA) and conductance-based (COBA) synapses (Section 3.6). We use the SBS python module (Section 2.2.2) to perform LIF sampling. The SBS version and parameters used for the neuron and synapse models are listed in Appendix A.2.2. The Tsodyks-Markram synapse model (TSO) (Section 3.7) is applied to maintain the height of the post-synaptic potentials as shown in Fig. 3.5. The used TSO parameters are listed in Table A.3. As a reference, we will regard Gibbs sampling (Section 3.3.2) which is the standard algorithm to generate samples in classical Boltzmann machines. To see whether mixing is important in the considered systems we will furthermore consider the adaptive simulated tempering (AST) sampling method (Section 3.3.3) which is especially designed to enhance mixing. The used AST parameters are listed in Table A.1.

4.1. Random Distributions

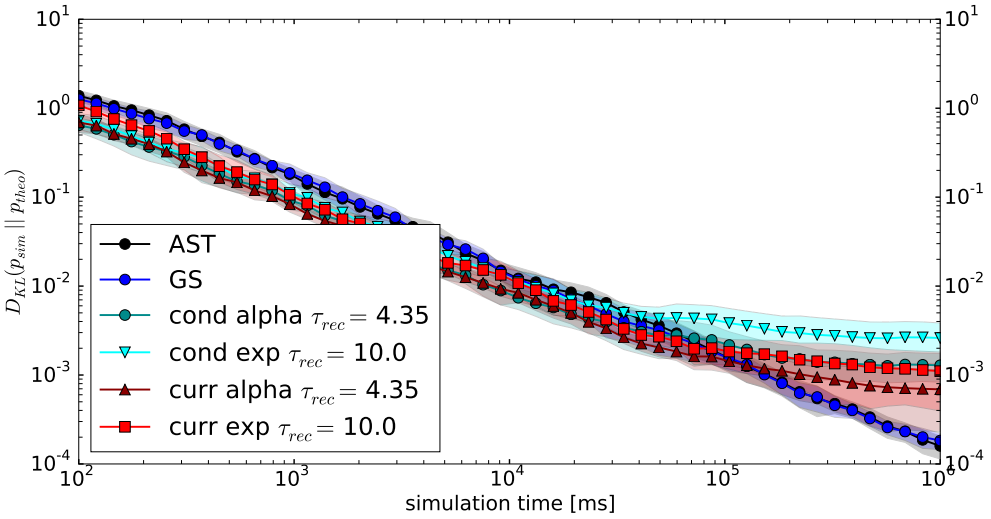
We start in this section with considering the behavior of the different sampling algorithms in BMs and RBMs where the weights and biases have been randomly chosen from the Gaussian distribution. To allow a deeper analysis we choose small systems which are still computationally tractable. The approach will be to calculate the theoretical Boltzmann distribution (3.10) according to the chosen weights and biases. Afterwards, we are sampling from the distribution with the above mentioned sampling algorithms. From the samples we calculate the sample distribution by determining the time (number of steps) spent in each state divided by the overall time (total number of steps). Finally, we evaluate the difference between the theoretical and sample distribution measured by the Kullback-Leibler divergence D_{KL} (Section 3.1) for different time points during sampling

to obtain a D_{KL} time evolution. We always evaluate the divergence from the sample to the theoretical distribution $D_{KL}(p_{\text{sim}}||p_{\text{theo}})$ to avoid having the sample distribution in the denominator in (3.1), because the probability of most of the states will be zero in the beginning. The theoretical Boltzmann distribution, in contrast, will never reach zero for finite weights and biases. In this section we will always run LIF sampling for 10^6 ms and compare it to 10^5 Gibbs and valid AST steps. However, it is not a cleared issue how to compare the sampling steps (time) for the different methods.
As mentioned in Section 3.3.3, we just count samples from the ground temperature level in AST. Hence, between two consequent valid AST samples there could be several samples from higher temperatures. The Wang-Landau algorithm in AST guarantees that each temperature level in AST occurs similarly frequent. Thus, as we have used ten temperatures levels (see Table A.1), we will in average count just every 10^{th} step. A direct comparison of each Gibbs step with an AST step seems therefore unfair. On the other hand, comparing every 10^{th} Gibbs steps with an AST step is also unfair because the dropped steps in AST are from a different distribution containing less information as it is flatter. We decided to compare within this thesis each Gibbs step with an AST step. As a consequence, the computational effort and therefore runtime for each AST step is roughly 10 times larger.
For comparing the LIF sampling time with Gibbs steps, we have a similar problem. How many seconds of simulation time should one choose to correspond to one Gibbs step? In this thesis we compare the refractory period of the LIF neurons ($\tau_{\text{ref}} = 10\,\text{ms}$) with one Gibbs step. This is due to the fact that the refractory period introduces an additional temporal correlation into the state evolution of the system which is of the order of the refractory time. However, it is questionable whether this is a fair comparison because the global state of the system changes just once in a Gibbs step but during 10 ms LIF simulation time it can change much more often.

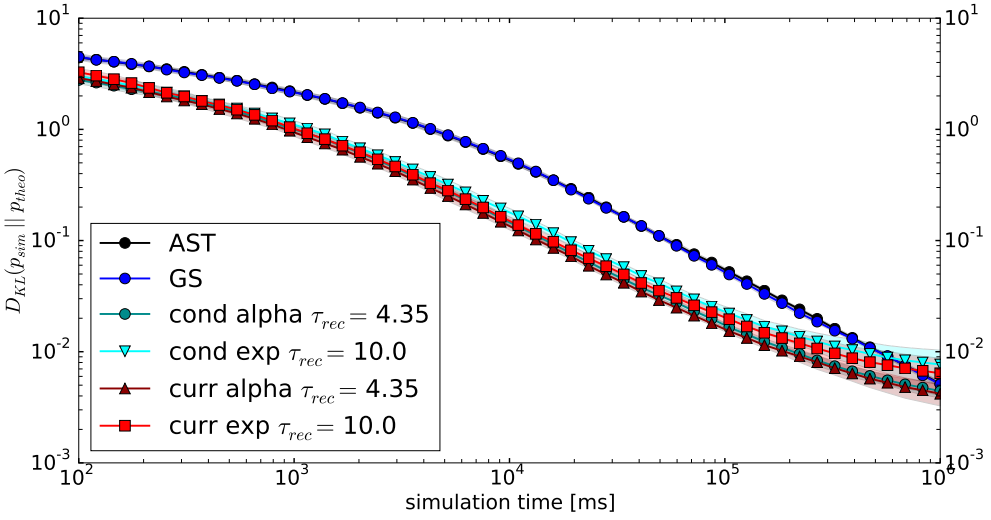
4.1.1. Homogeneous Distributions

In the following we assume that we have fully connected BMs where the weights and biases are Gaussian distributed around $\mu = 0$ with a small variance of $\sigma = 0.3$, which will lead to quite homogeneous distributions. For these distributions we expect that every sampling algorithm will work well. Hence, these systems are suited to investigate the general properties of the considered sampling algorithms.
To test the algorithms we have applied the above mentioned approach. From this we obtain the D_{KL} time evolutions shown in Fig. 4.1 for fully connected BMs with 5, 10, 15 and 20 units. The D_{KL} values at the end of the simulation are collected in Table A.10 and can be used as reference results for the different sampling algorithms in this standard case. In Table A.11 we collected analogous results with varying seeds for the sampling algorithms.
In Fig. 4.1a one can see that all methods seem to work fine in the beginning as the difference between the theoretical and sample distribution measured by the D_{KL} becomes smaller over time (sampling steps). In theory we would expect a straight line in the double logarithmic plot. For Gibbs and AST this is true but for LIF sampling we see a saturation

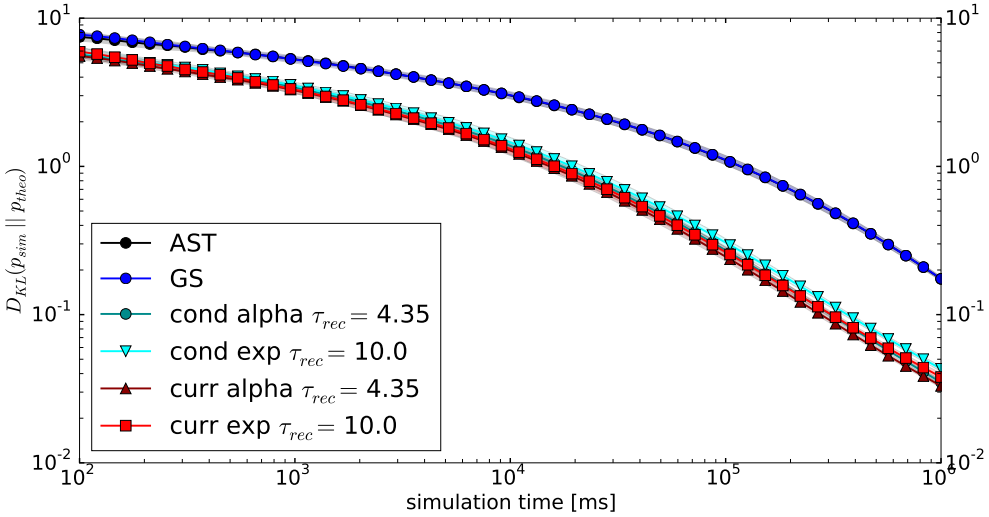
Figure 4.1.: D_{KL} time evolution for BMs



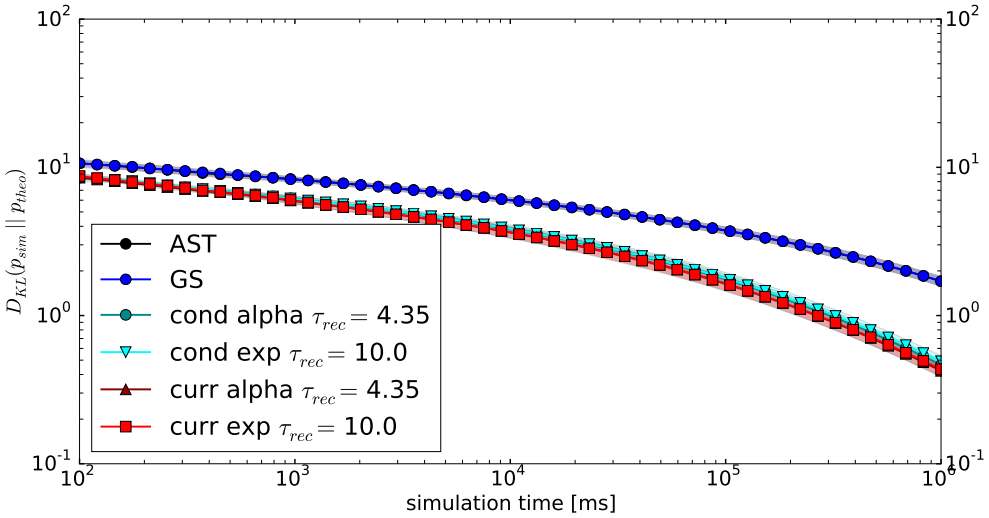
(a) 5 units



(b) 10 units



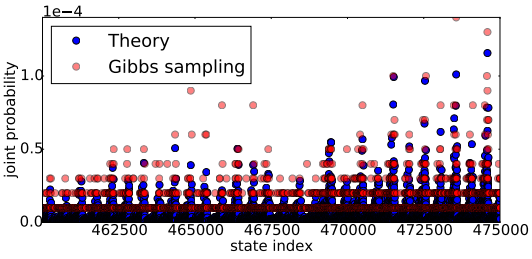
(c) 15 units



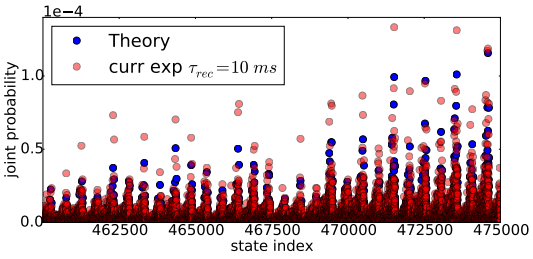
(d) 20 units

Figure 4.1.: Time evolution of the D_{KL} between the theoretical and sample joint distribution for fully connected BMs with 5, 10, 15 and 20 units. The weights and biases have been drawn from a Gaussian with $\mu = 0$ and $\sigma = 0.3$. The sample distribution are created from Gibbs (GS) sampling, AST and LIF sampling with both CUBA (curr), COBA (cond) and exponential (exp) and alpha-shaped synapses. τ_{rec} is the recovery time constant used for TSO. It is chosen such that the post-synaptic potential (PSP) height is maintained (see Fig. 3.5). The displayed D_{KL} values denote the mean over ten tries where we have drawn new weights and biases for each try. The colored area shows the standard deviation.

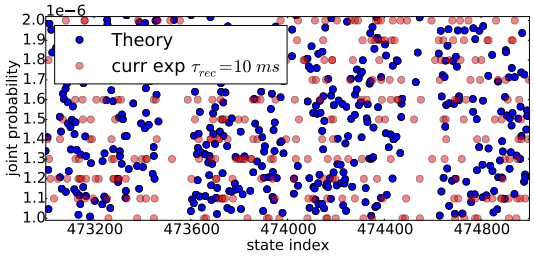
at the end. This is due to the systematic error caused by the different PSP shape in LIF sampling compared to the rectangular PSP which is required in theory, as illustrated in Fig. 3.7. Considering the different LIF sampler one can see that they perform similarly. Just in the end they tend to settle on different saturation levels. However, regarding the standard deviation this effect is not significant.
When the size of the BM is increased, we can see that the D_{KL} values become worse. This is because the number of possible states whose probabilities need to be matched rises exponentially but the time in which the states can be populated during sampling stays the same. Hence, we get a much rougher approximation of the sampling probabilities for each single state. This effect is especially serious in the beginning which makes the shape deviate from the expected straight line. The errors become also smaller because for such homogeneous distributions nearly every chosen state will be a new state in the beginning and the state probabilities are similar. As a consequence, they all lead to a similar improvement in the D_{KL} calculation, which leads to a similar behavior of the sampling algorithms for different tries.
Another effect visible for larger BMs is that LIF sampling performs better than the abstract samplers. Even for the 5 unit BM in Fig. 4.1a this can be observed in the beginning. This is due to our choice of comparing one Gibbs step with 10 ms of LIF sampling time. The consequences can be seen when looking at the joint distribution especially for larger systems. In Fig. 4.2 a cutout of the joint distribution for the 20 unit BM is shown. We see that Gibbs exhibits just a rather discretized approximation of the theoretical distribution. This is because we have in total $2^{20} \approx 10^6$ possible states which occur with similar probabilities in homogeneous distributions. Hence the typical probability values vary around the value of 10^{-6}. However, we obtained this plot after 10^5 sampling steps. Thus, when we sample during the whole run just once from a certain state its probability would be already 10^{-5}. The minimal resolution of the joint probability with Gibbs sampling is consequently one order higher as the typical theoretical values.
On the other hand, in Fig. 4.2b we can see that LIF sampling is more precise as we have a spike time resolution and therefore a possibly new state every 0.1 ms. With the total simulation time of 10^6 ms this leads to a resolution of 10^{-7}. Hence, the LIF resolution becomes visible when we zoom further into the distribution as demonstrated in Fig. 4.2c. This shows that, comparing one Gibbs step with 10 ms, LIF sampling can approximate high-dimensional distributions faster than the abstract sampling algorithms.
Our aim later will be to train large RBMs. However, due to the exponentially increasing state space, it is impossible to calculate the theoretical joint distribution and therefore the D_{KL} for the full system. To still get an impression how well the sampled distribution approximates the theoretical one, one can exploit the bipartite structure of RBMs. We will therefore consider next RBMs where the visible layer is large and the hidden layer small. With the simplified calculation of the hidden distribution in (3.16) we can at least observe the behavior for the hidden layer which represents the features of the visible layer.
In the following we will therefore consider similar results as before but now with RBMs with 100 visible units and 5, 10 , 15 and 20 hidden units. We evaluate the D_{KL} values



(a) Gibbs sampling



(b) LIF sampling



(c) LIF sampling zoom

Figure 4.2.: Cutout of the joint theoretical and sample distribution for a BM with 20 units for one of the runs in Fig. 4.1d. On the x-axis the state index is plotted. Its number converted to a binary string of length 20 describes which units are active. The height of the dots over the indices shows the probability of the state belonging to the index.
Figure 4.2a shows the comparison between the sample joint distribution from Gibbs sampling and the theoretical one. In Fig. 4.2b the corresponding result for LIF sampling with CUBA exponential-shaped synapses is shown. Figure 4.2c is a zoomed in part of Fig. 4.2b.

between the theoretical hidden distribution calculated with (3.16) and the corresponding sampled hidden distribution. The final D_{KL} values are collected in Table A.12. To see the influence of the simulation seed the same results are collected in Table A.13 with varying seeds for the sampler. Figure 4.3 shows the result of the D_{KL} time evolution for the 5 hidden units example. We omit the plots with more hidden units because they exhibit exactly the same tendency as in the BM case before. In Fig. 4.3 there is, as for normal BMs, a straight line for Gibbs, AST and LIF sampling in the beginning until it saturates. However, compared to the BM result Fig. 4.1a, the saturation happens on higher D_{KL} levels. This is probably due to the larger size of the system. As each hidden neuron is connected to 100 visible ones it can receive much more spikes per time step than in the small 5 unit BM. This leads to a summation of the different PSP shapes, which causes a more significant deviation, despite the small weights. The differences between the LIF sampling algorithms are again not significant compared to their standard deviation.

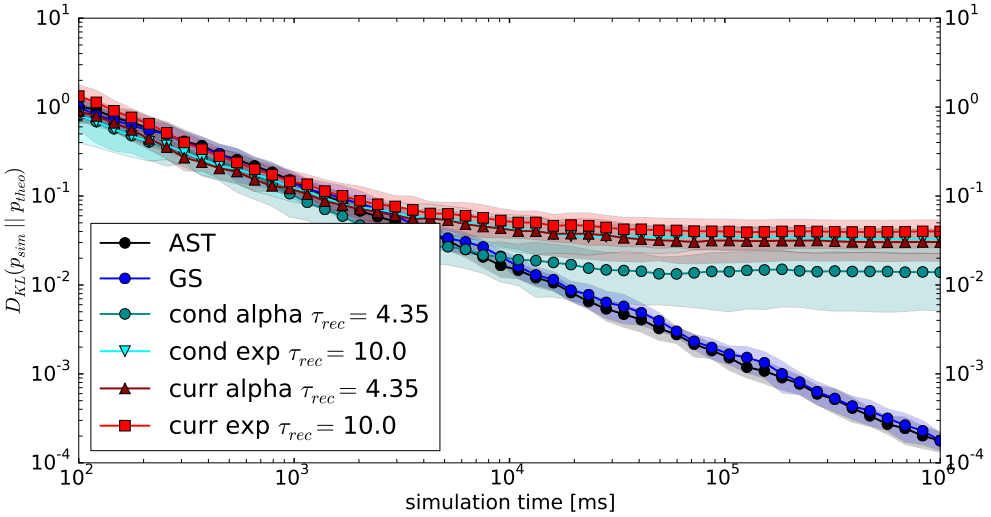


Figure 4.3.: Time evolution of the D_{KL} value between the theoretical and sampled joint hidden distribution for an RBM with 100 visible and 5 hidden units. The cases with 10, 15 and 20 hidden units show a similar behavior as in Fig. 4.1 and are therefore omitted. The weights and biases have been drawn from a Gaussian with $\mu = 0$ and $\sigma = 0.3$. The sampled distributions are created from Gibbs (GS), AST and LIF sampling with both CUBA, COBA and exponential, alpha-shaped synapses. τ_{rec} is the recovery time constant used for TSO. It is chosen such that the PSP height is maintained (see Fig. 3.5) The displayed D_{KL} values denote the mean over ten tries where we have drawn new weights and biases for each try. The colored area shows the standard deviation.

Furthermore, we can see that there is no significant difference between Gibbs and AST. Hence it seems that even for the larger RBMs the mixing issue plays no role.
We conclude from this that increasing the size of the system alone makes mixing not important yet. We probably need to consider more inhomogeneous distributions via choosing weights and biases with a larger variance, which will be considered in the next chapter. Moreover, we see that due to the discretization effect it makes no sense to compare joint or hidden distributions for more than 10 units.

4.1.2. Inhomogeneous Distributions

The aim in this section will be to find out what happens if we choose the weights and biases with large variances leading to inhomogeneous distributions, where some states appear with a high probability while most of the others have a nearly vanishing one. In these systems one could expect that mixing plays a more important role. To create such systems we will in the following again consider RBMs with 100 visible and 5 hidden units.

This time, however, we draw the weights and biases from a Gaussian with mean $\mu = 0$ and a higher standard deviation of $\sigma = 1.5$. Otherwise the procedure will be exactly the same as in the previous section. For the D_{KL} time evolution of the hidden distribution we obtain Fig. 4.4

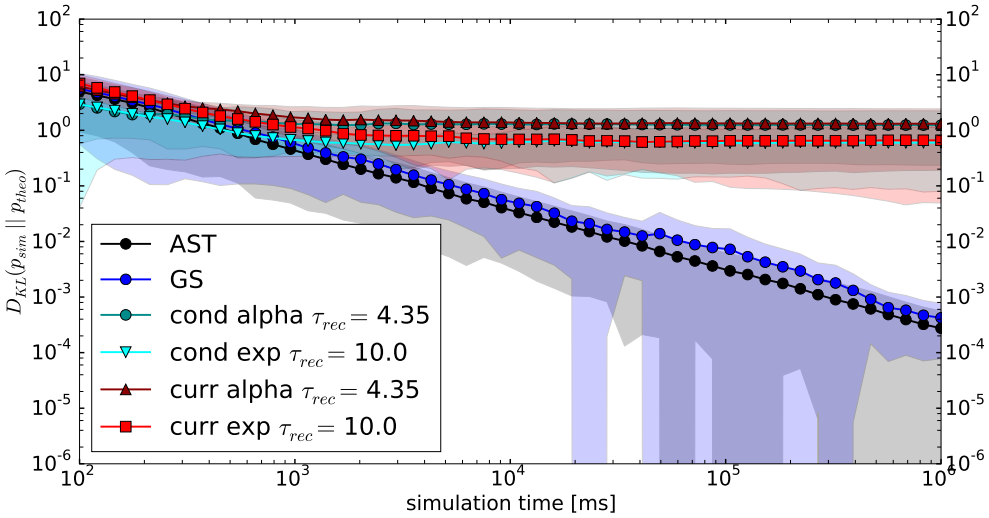
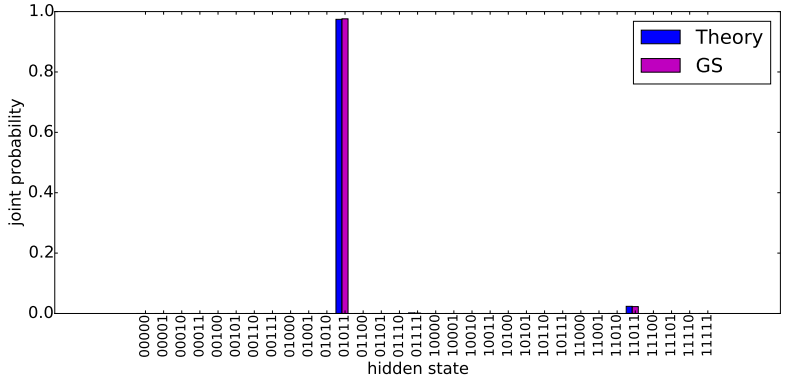
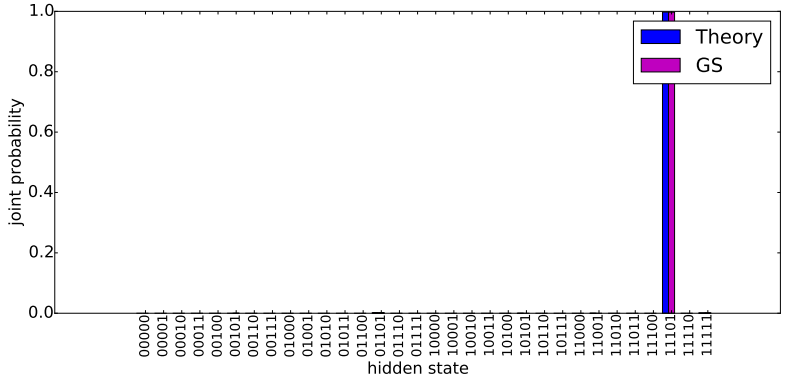


Figure 4.4.: Time evolution of the $D_{KL}(p_{\text{sim}}||p_{\text{theo}})$ between the sample distributions p_{sim} for the different sampling algorithms and the theoretical distribution p_{theo}. Depicted is the mean of the D_{KL} values over ten tries where for each one a new set of weights and biases have been chosen. The colored area illustrates the standard deviation. The weights and biases have been chosen from a Gaussian with $\mu = 0$ and $\sigma = 1.5$.

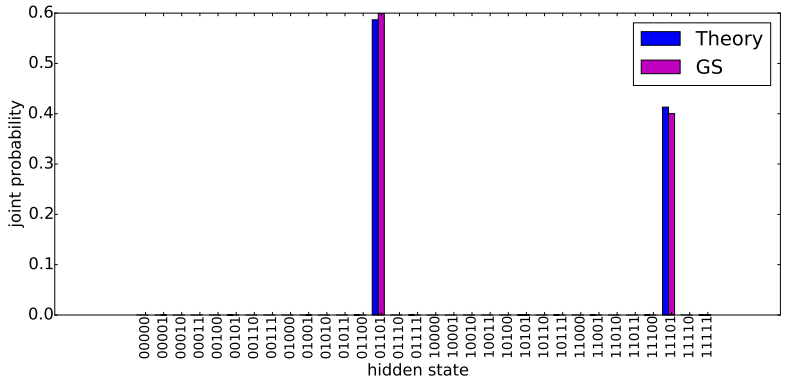
We can see that compared to the corresponding simulation for homogeneous distributions in Fig. 4.3 all sampler exhibit larger standard deviations. Furthermore we see that Gibbs sampling and AST perform similarly good as in the homogeneous case. LIF sampling, however, has a much worse performance. To explain these results we take a look at some examples of the joint hidden distributions at the end of the simulation time.
Figure 4.5 shows three examples for Gibbs sampling. We see that the chosen weights and biases indeed lead to very inhomogeneous distributions, but the Gibbs sampler still manages to reproduce them very well. Beside that, we see that many different distributions are possible. Some where just one state appears with a probability of nearly one and others in which we have several modes. This explains the larger variations in the D_{KL} values. However, this is also a systematic problem of this approach to create systems where mixing is important. For example in Fig. 4.5b mixing would be even a disadvantage because the whole probability mass is concentrated in just one state. On the other hand in Fig. 4.5c a better mixing would probably be an advantage. One can even see that this is the only system where Gibbs exhibits a small deviation from the theoretical distribution. For an ideal system where mixing is important we would expect



(a) Gibbs sampling example 1



(b) Gibbs sampling example 2



(c) Gibbs sampling example 3

Figure 4.5.: Three examples of the joint hidden distribution each comparing the theoretical distribution with the one obtained from Gibbs sampling. The examples are taken from the 10 runs in the D_{KL} time evolution in Fig. 4.4. On the x-axis all possible state combinations for the hidden layer are shown. The y-axis shows the probabilities for them to appear.

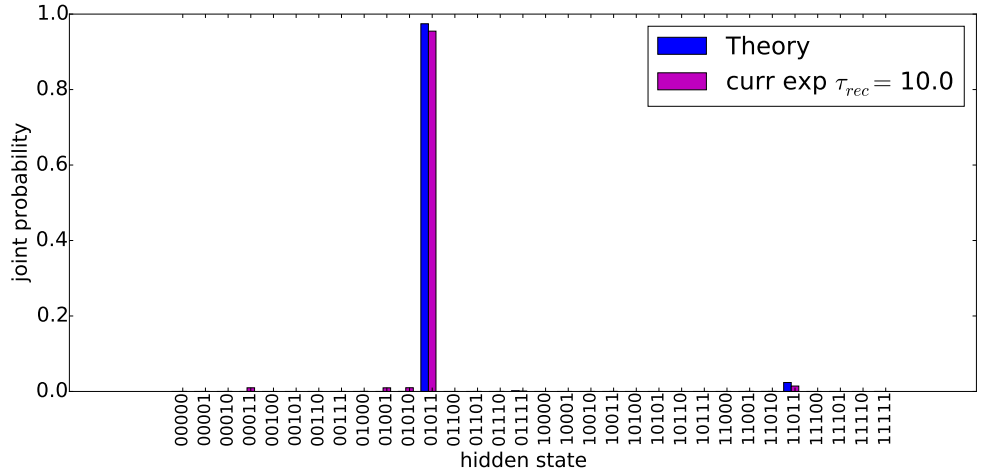


Figure 4.6.: Example of the joint hidden distribution comparing the theoretical distribution with one obtained from LIF sampling with CUBA exponential-shaped synapses. The example is taken from the first of the 10 tries in the D_{KL} time evolution in Fig. 4.4.

a distribution with several dominant modes lying roughly at the same level, but with this approach it depends on chance whether we obtain such a system.
To understand why LIF sampling leads to such bad results we further consider in Fig. 4.6 an example of the hidden distribution for LIF sampling as well. One can see that the distribution here is quite similar to the corresponding one for Gibbs sampling in Fig. 4.5a. The LIF sample distribution, however, shows some tiny additional peaks in the state space for similar states as the dominant one (each vary just in the state of one unit). These additional states appear for all LIF sampler. They could be caused by deviations of the LIF activation function (see Section 3.8), the different PSP shape compared to abstract sampling or the delay time of 0.1 ms with which the information that a neuron spiked reaches the others. Although these states appear with a small probability, they have regarding (3.1) a strong influence on the D_{KL} value because the corresponding theoretical probability for these states is nearly zero. This leads to the bad performance of the LIF sampler in the D_{KL} time evolution in Fig. 4.4.
Altogether, we learn from the results in this section that drawing weights and biases from a completely random distribution with large variances leads not to the desired results. The resulting distributions are unpredictable and mixing seems to play in most cases no important role. We can only hope to find by chance a distribution where mixing is important, which makes a systematic treatment difficult. This result led us to the approach in the next section, where instead of drawing weights and biases randomly we set them manually to form certain patterns which we have devised in advance.

4.2. Multimodal Distributions with Artificial Patterns

Our approach in this section is to manually imprint certain patterns in Boltzmann machines by choosing the weights and biases accordingly. We use solely RBMs for this because here one can easily interpret the weights by looking at the receptive fields of the hidden units (see Section 3.2.1). In this section we first describe the way how we create the patterns. Afterwards, we look at an example of the sampling performance for the created patterns. Finally, we investigate how this performance behaves when we vary the system size and the strength of the imprinted patterns.

4.2.1. Pattern Creation

To create multimodal artificial patterns we use an RBM with four hidden units and imprint four bar patterns into the weight connections between visible and hidden units as illustrated in Fig. 4.7. The values of the weight connections W_{exc} and W_{inh} determine the strength of the imprinted pattern. Their influence will be investigated in Section 4.2.4. Further parameters are the size of the visible layer and the biases of the units. The influence of the system size will be investigated in Section 4.2.3. For the biases we choose for simplicity the same value of $B = -1$ for all units.

4.2.2. Mixing for the Artificial Multimodal Pattern

In this section we consider the mixing behavior of an example of a pattern created with the approach in Section 4.2.1. We choose symmetric weight connections of $W_{\text{exc}} = 1.2$ and $W_{\text{inh}} = -1.2$. The size of the visible layer is 10x10. Due to the symmetry of the weights, the same bias for all units and the fact that the bars fill up exactly the half of the visible layer, we will obtain a symmetric distribution where the modes for the four bars appear with exactly the same probability.
As in Section 4.1 we sample from such a distribution and consider the $D_{KL}(p_{\text{sim}}||p_{\text{theo}})$ time evolution, which yields to Fig. 4.8. We can see that just the AST sampler approaches the theoretical distribution with the expected straight line. Gibbs sampling shows no significant convergence and the LIF sampler saturate early on a high level. One can see that LIF sampling with alpha-shaped synapses seems to have a slight advantage compared to LIF sampling with exponential-shaped synapses while switching between CUBA and COBA has no significant influence. The fact that AST performs so much better than the other sampling algorithms indicates that in this system mixing is indeed an important issue. To explain the D_{KL} results we take a look at some examples for hidden distributions for the abstract sampler (Fig. 4.9) and LIF sampling (Fig. 4.10).
We can see in Fig. 4.9a that Gibbs sampling stays only in one mode. This explains why its D_{KL} value in Fig. 4.8 did not change over time. The slight convergence in the end in Fig. 4.8 indicates that in a few of the ten runs Gibbs sampling managed to reach other modes.
In contrast to this we see in Fig. 4.9b that AST is able to mix between the four modes and therefore reproduces the theoretical distribution very well. This leads to its small

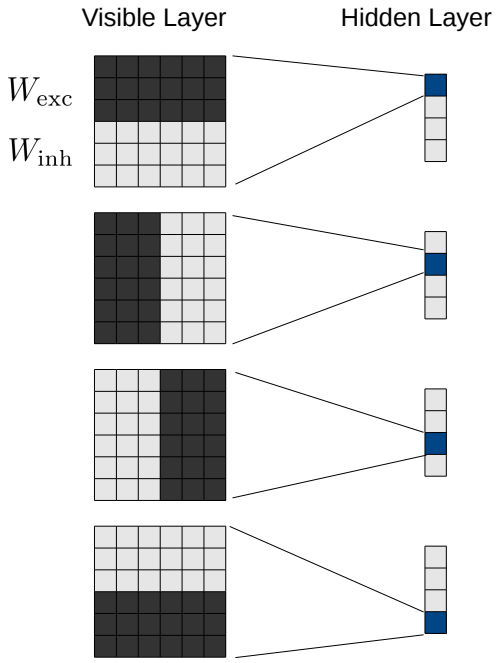


Figure 4.7.: Scheme of the creation of artificial bar patterns. We use 4 hidden units and imprint manually bar patterns into the weight connections for each of them. The four imprinted bars are shown in the visible layer of the scheme. They correspond to the receptive fields of the hidden units. Each bar fills half of the visible layer. The dark boxes stand for a positive weight connection between visible and hidden units. They have all the same value of W_{exc}. The light gray boxes stand for a negative connection W_{inh}.

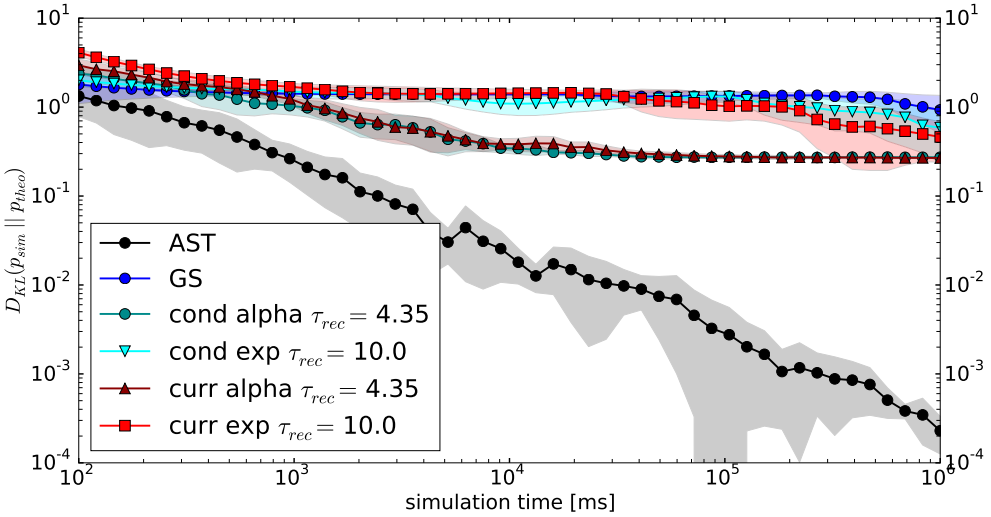
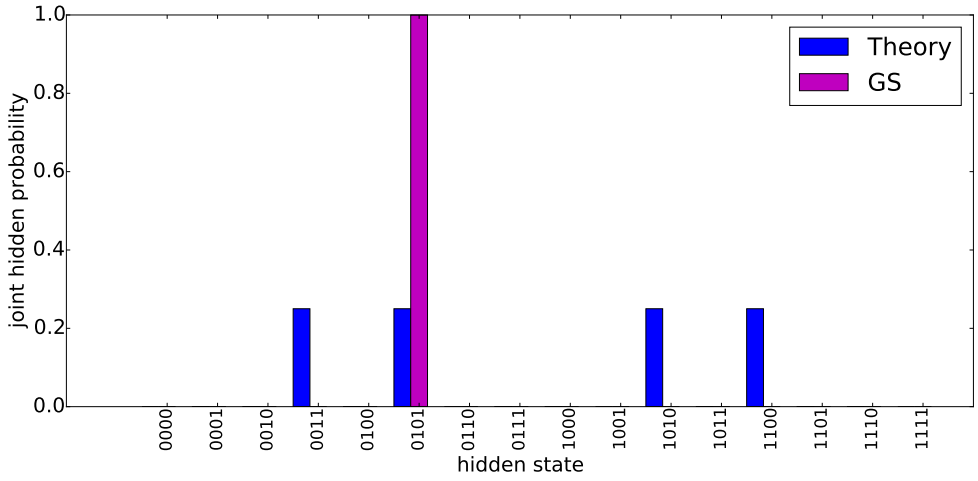
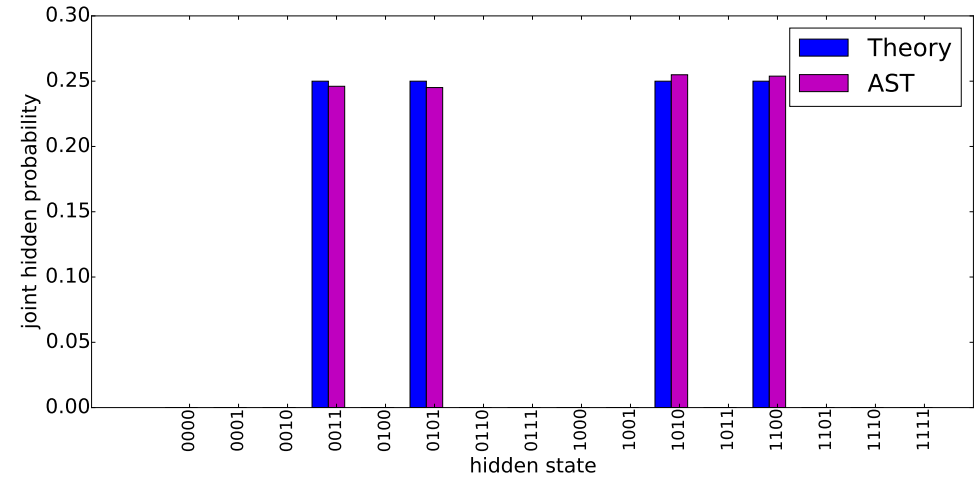


Figure 4.8.: Time evolution of the mean D_{KL} values for the 4 bar pattern with $W_{\text{exc}} = 1.2$ and $W_{\text{inh}} = -1.2$. The size of the visible layer is 10x10 and the bias is $B = -1$ for every unit. We calculate the mean over 10 tries in which we vary the seed for the random number generator in the sampling algorithms. The colored area represents the standard deviation.

D_{KL} value.
Figure 4.10 reveals that LIF sampling mixes in case of exponential-shaped synapses a little and in case of alpha-shaped synapses much better than Gibbs sampling. The reason for the still quite bad D_{KL} values can be seen in the hidden distribution plots in Fig. 4.10. Both LIF sampler go into some states where the theoretical probability is nearly vanishing. This is similar to the results for the inhomogeneous distributions in Section 4.1.2. Considering the D_{KL} formula (3.1) these additional states have a huge contribution to the sum, which leads to the bad performance in the D_{KL} time evolution for the LIF sampler.
To obtain a better measure for this case which respects just the mixing behavior in the dominant modes we consider a reduced D_{KL} measure.
For this measure we sum in (3.1) just over the contributions of the four dominant modes. This has the advantage that the small additional states in LIF sampling, which are unavoidable, do not effect the D_{KL} values. However, in Section 3.1 we have seen that not summing over all possible states of the distributions makes it possible to have negative D_{KL} values. As a consequence, we need to carefully treat the results for the reduced D_{KL} because a reduced D_{KL} of zero does not necessarily mean a perfect match of the distributions anymore. Deviations in different directions can cancel each other and therefore also lead to a reduced D_{KL} of zero. As negative and positive reduced D_{KL} values both represent deviations we take the absolute value of the sum in (3.1) to make it still possible to show the results in a double logarithmic plot. The results in Fig. 4.11 show

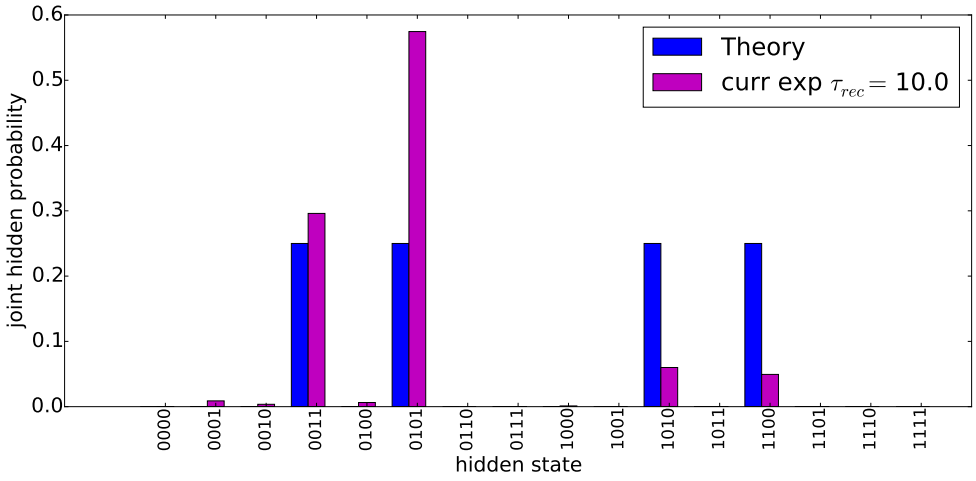


(a) Gibbs sampling

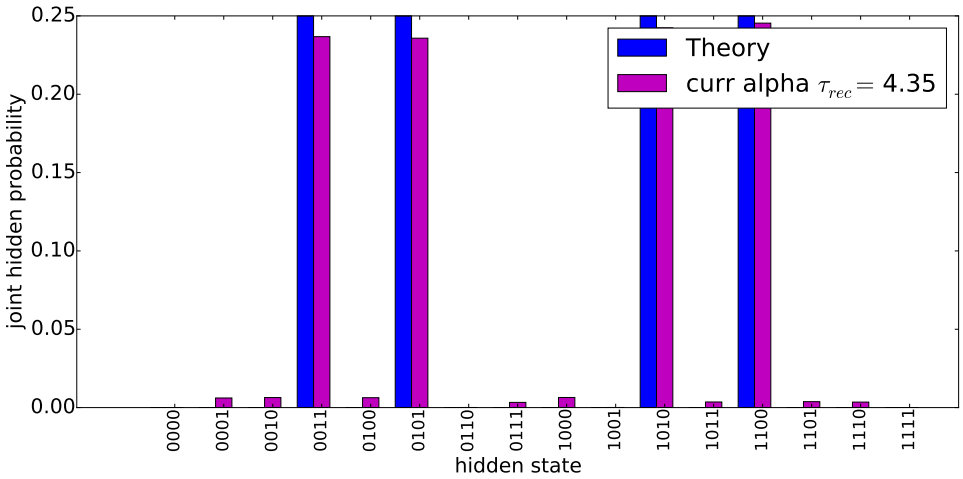


(b) AST

Figure 4.9.: Examples for hidden distributions in the 4 bar example for the abstract sampling algorithms Gibbs and AST. These show the final result of one run in Fig. 4.8. On the x-axis all 2^4 possible combinations of the hidden layer are listed. The y-axis shows the probabilities for them to appear.



(a) LIF sampling with exponential-shaped synapses



(b) LIF sampling with alpha-shaped synapses

Figure 4.10.: Examples for hidden distributions in the 4 bar example for the LIF sampling methods with CUBA exponential- and alpha-shaped synapses. They correspond to the plots for the abstract samplers in Fig. 4.9. The corresponding COBA methods perform similarly and are therefore omitted.

that the performance for alpha-shaped LIF sampling is indeed much better than for Gibbs sampling. Exponential-shaped LIF exhibits just a marginally improved performance. This is in accordance to the examples for the hidden distributions shown in Fig. 4.10. Hence, in this case, the reduced D_{KL} seems to produce reasonable results.

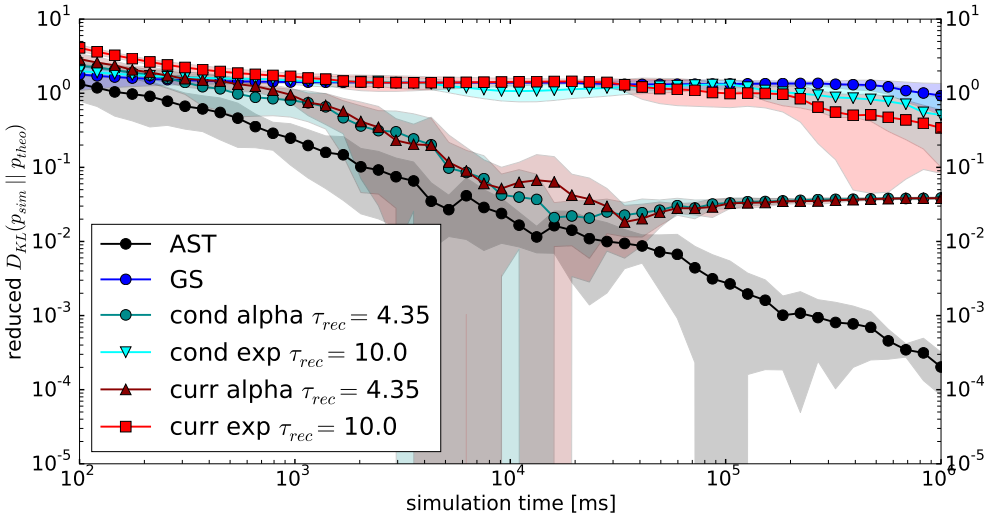
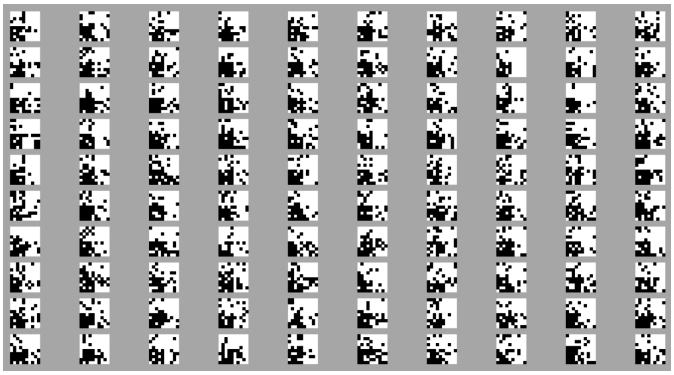
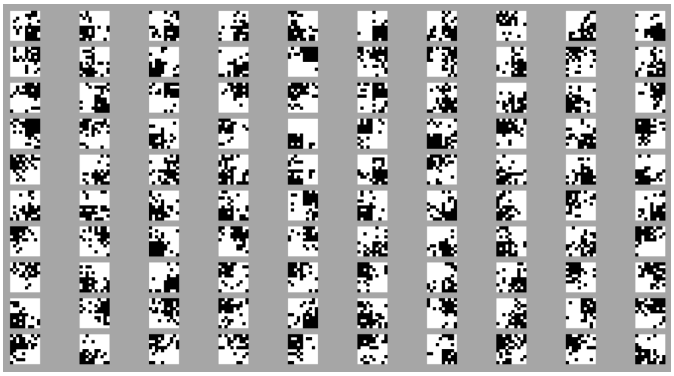


Figure 4.11.: Time evolution of the reduced D_{KL} values for the 4 bar example corresponding to Fig. 4.8.

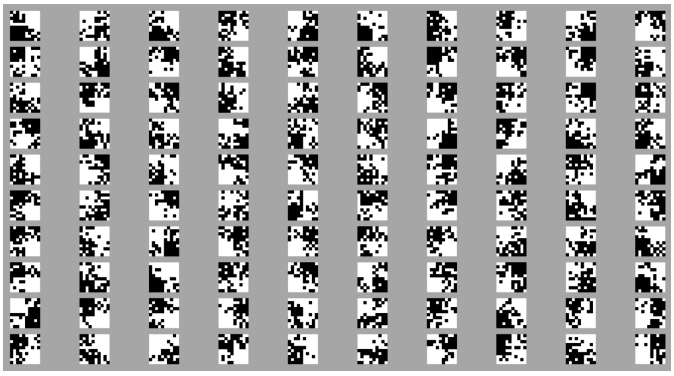
Until now we had just observed the behavior of the hidden units. However, it is not guaranteed that the visible units behave as we would expect it. To make sure that this is the case we consider, in the following, examples of the produced samples of the visible layer in 10x10 format. The hidden distributions in Fig. 4.9 and Fig. 4.10 reveal that the dominant modes belong to states where two hidden neurons are active. Keeping the imprinted bar patterns (Fig. 4.7) in mind, we conclude that these states correspond to two superimposed bars in each of the four edges.
We therefore expect sampled images where one edge is dark, because it is activated by two bar patterns. The two neighboring edges should exhibit an intermediate activation because they are activated by just one bar respectively. The remaining edge should be mainly deactivated.
Examples for the sampled images are shown in Fig. 4.12. We indeed see the expected pattern but with some noise added which is usual for sampling. Hence the visible states follow the behavior of the hidden states. This becomes also obvious considering the results for the different sampler. For Gibbs sampling (Fig. 4.12a) we see that, as indicated by the joint hidden distribution in Fig. 4.9a, the lower left edge is active for the whole duration because it is not able to mix. For AST, however, we see in Fig. 4.12b that the sampler produces samples from all edges and has no problems with mixing. The same is true for LIF sampling (Fig. 4.12c).
This example shows that for a particular choice of weights and biases in the bar pattern



(a) Samples produced by Gibbs sampling



(b) Samples produced by AST



(c) Samples produced by LIF sampling with alpha-shaped synapses

Figure 4.12.: Samples of the visible layer produced by Gibbs sampling, AST and LIF sampling with CUBA alpha-shaped synapses. The visible units are arranged in 10x10 images as the imprinted bar patterns. The black and white dots denote units which are activated and deactivated respectively. For each sampler 100 samples taken equally distributed over the sampling time are shown. Hence there are 10^4 ms between two LIF samples and 10^3 steps between two Gibbs/AST samples. The samples should be read from left to right and up to down. They are taken from the same run as the hidden distribution plots in Fig. 4.9 and Fig. 4.10.

scheme, we indeed found a system where mixing is a critical issue. Here we saw that Gibbs was not able to reproduce the theoretical distribution while AST fixed this problem. LIF sampling yielded an intermediate performance. To make sure that this was not just a lucky choice of parameters, we will in the following sections investigate when mixing starts to become important by varying the system size and the strength of the imprinted patterns.

4.2.3. Influence of System Size

In this section we investigate how the sampling behavior for the bar pattern scheme described in Section 4.2.1 changes with the size of the visible layer. For the imprinted patterns we use the same weights and biases as in Section 4.2.2. We now determine the mean final D_{KL} values for an increasing number of visible units. The result is shown in Fig. 4.13.

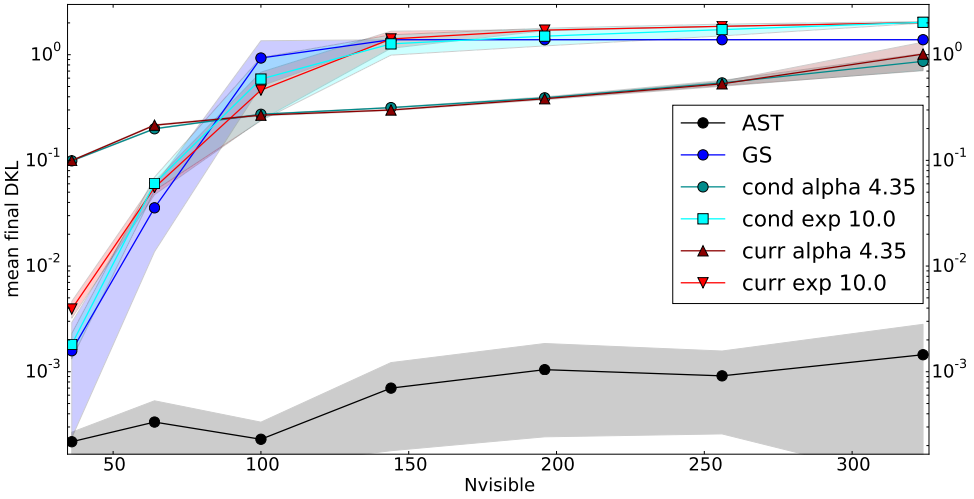


Figure 4.13.: Mean final D_{KL} values after 10^6 ms simulation time for LIF sampling and 10^5 steps for Gibbs sampling and AST. The x-axis shows the size of the visible layer. As the visible layer is interpreted as a two-dimensional image we vary the size from 6^2 to 18^2. The mean is calculated over ten trials with varying seeds for the random number generator for the sampling algorithms. The weights and biases are the same as in Section 4.2.1.

We see that as expected for the case with 100 visible units we get the same final result as in Section 4.2.2. Increasing the RBM further leads to a similar outcome. AST is still able to reproduce the distribution. Gibbs and now also LIF sampling with exponential-shaped synapses do not mix at all. LIF sampling with alpha-shaped synapses leads to a slightly better performance. In Section 4.2.2 we have already seen the reason for the relative high D_{KL} values for alpha-shaped LIF sampling. It is due to additional states which appear

for the LIF sampler but should be in theory nearly zero. To avoid the strong influence of these states we again consider the reduced D_{KL} (Fig. 4.14). The lower D_{KL} values for LIF alpha-shaped sampling confirm that it is still able to mix between the four bars. Its ability to mix starts to break down only for very large systems. The fact that it becomes better in the middle with increasing number of visible units is an artifact of the reduced D_{KL}, where negative and positive deviations cancel each other.

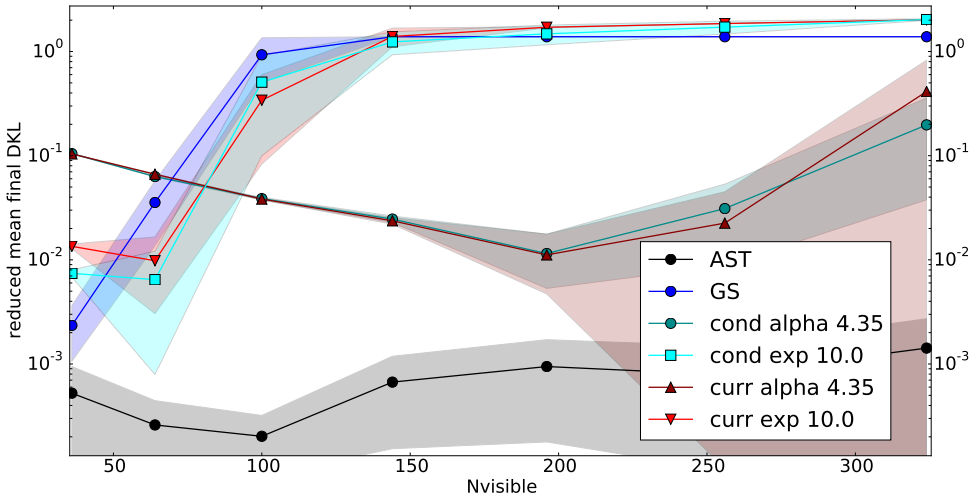
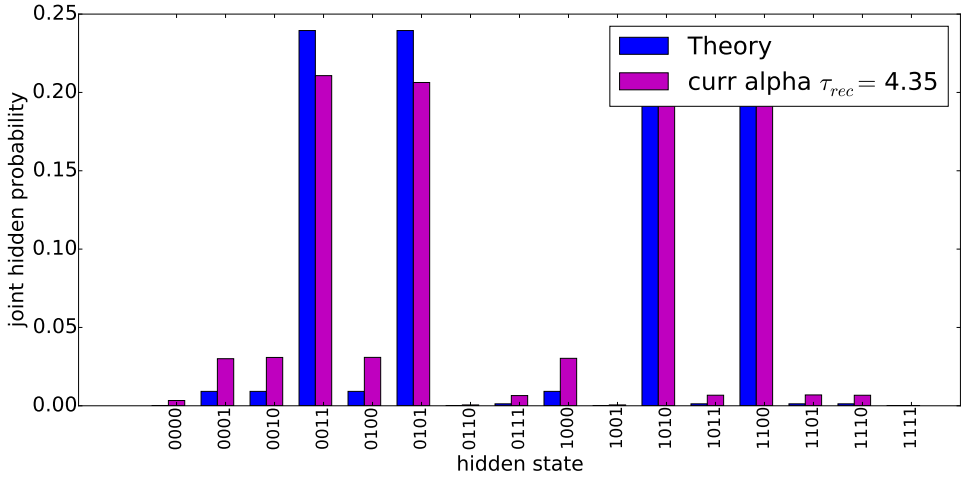
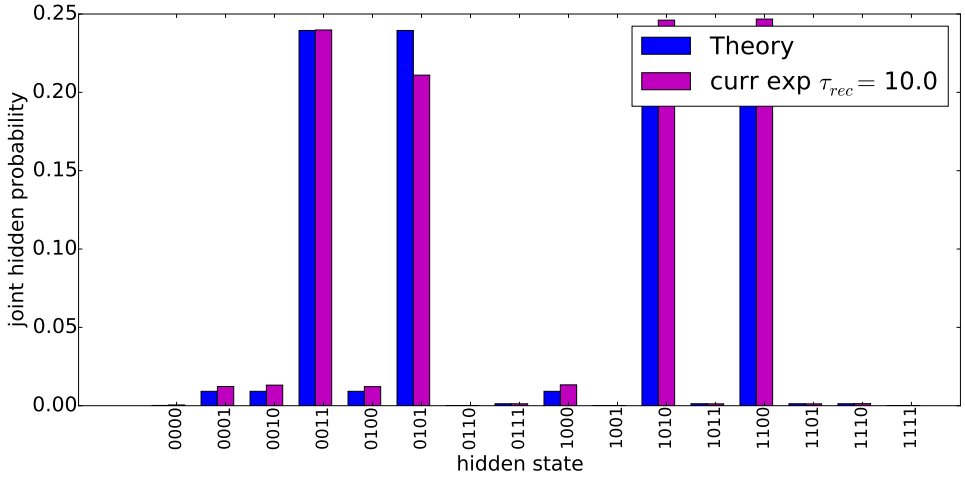


Figure 4.14.: Mean final reduced D_{KL} values corresponding to Fig. 4.13.

For small systems sizes one can see in Fig. 4.14 that Gibbs and LIF exponential-shaped performs better than alpha-shaped. This is because alpha-shaped LIF tends to have larger additional states then exponential-shaped. This makes it perform worse for smaller systems as can be seen in Fig. 4.15. These results demonstrate that the example which we discussed in Section 4.2.2 can be generalized to different system sizes. When the system has more than 64 visible units mixing becomes an important issue, which can be seen by the fact that Gibbs and exponential-shaped LIF sampling begin to stop mixing in this case. Alpha-shaped LIF sampling, however, is still able to mix, but only on the cost of producing additional states.



(a) LIF sampling with exponential-shapes synapses



(b) LIF sampling with alpha-shaped synapses

Figure 4.15.: Examples for the hidden distributions in the 4 bar example with 6^2 visible units. Depicted are the results for the LIF sampling methods with CUBA exponential- and alpha-shaped synapses. The corresponding COBA methods perform similarly and are therefore omitted.

4.2.4. Influence of Pattern Strength

In this section we investigate how the sampling behavior for the bar pattern scheme described in Section 4.2.1 changes with the strength of the imprinted patterns. We

choose the same system size of 4 hidden units and 100 visible units as in Section 4.2.2. The mean final D_{KL} values are this time determined for an increasing value of W_{exc}. To keep the patterns symmetric we use $W_{\text{inh}} = -W_{\text{exc}}$. The biases remain unchanged. The result is presented in Fig. 4.16.

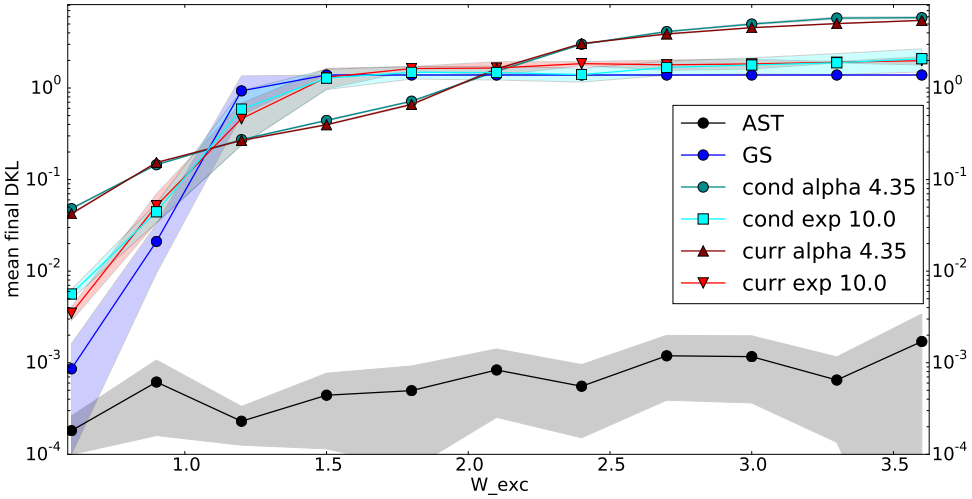
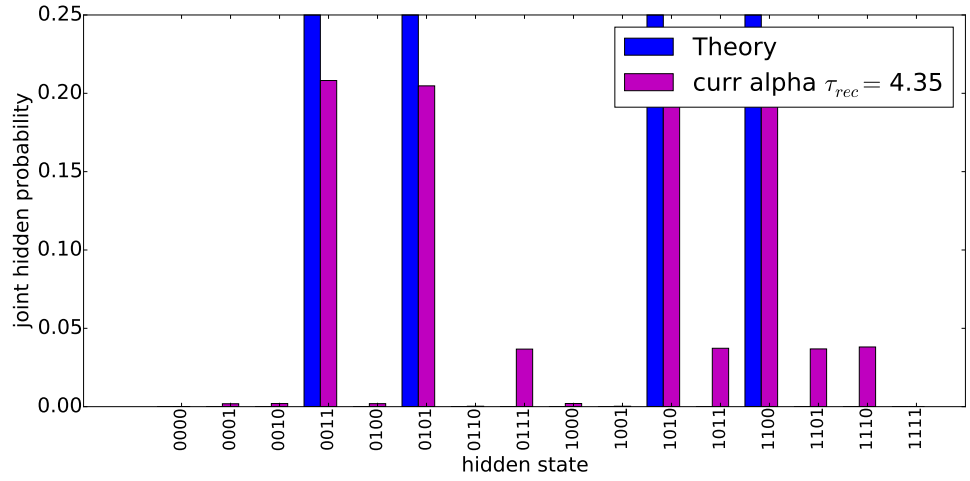
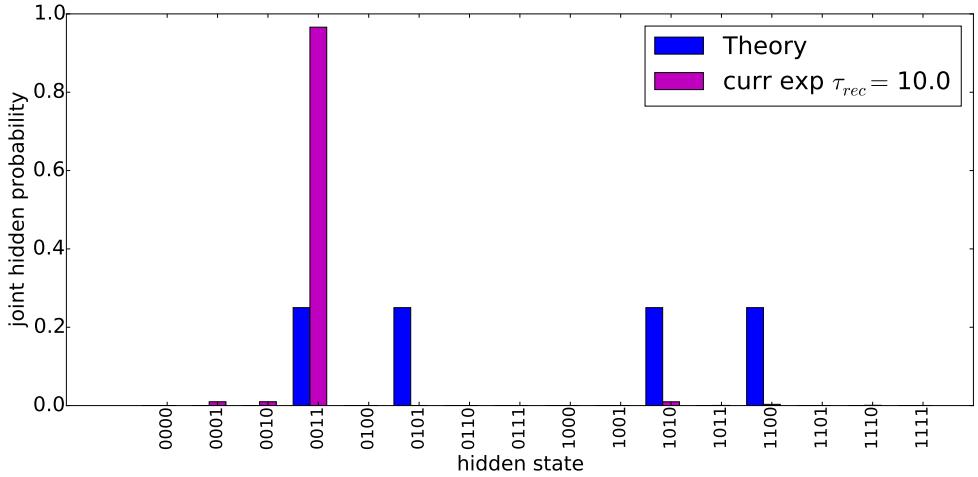


Figure 4.16.: Mean final D_{KL} values after 10^6 ms simulation time for LIF sampling and 10^5 steps for Gibbs sampling and AST. On the x-axis the positive weights of the patterns W_{exc} is increased. For the negative weights we chose $W_{\text{inh}} = -W_{\text{exc}}$. The mean is calculated over ten trials with varying seeds for the random number generator for the sampling algorithms.

One can see that, as expected, for small weights all sampler seem to perform the best. The LIF sampler are worse than the abstract sampler which is due to the systematic error which we have seen in Section 4.1.1 for the random distributions before. The difference between CUBA and COBA is again insignificant for all weights but LIF with alpha-shaped synapse seems to perform worse in the beginning. The reason for this is the same as in Fig. 4.14. With increasing W_{exc} the performance for all samplers except of AST becomes worse. LIF sampling with alpha-shaped synapses becomes better than exponential-shaped for intermediate weights but settles on a high value for large weights. However, considering the hidden distributions for LIF sampling for the largest case of $W_{\text{exc}} = 3.6$ in Fig. 4.17 reveals that it is still mixing while LIF sampling with exponential-shaped synapses stays just in one mode. Therefore, we will consider again the reduced D_{KL} in Fig. 4.18. It shows us that, in terms of mixing between the four bars, alpha-shaped LIF sampling indeed produces intermediate results even for high weights. The hidden distributions show in all tries similar results as in Fig. 4.17b. We see especially that the sampled probabilities for the four dominant modes are always smaller than their theoretical counterparts because a significant part of the sampled probability mass is lost in the additional states. As a consequence, we can be sure that this time the reduced



(a) LIF sampling with exponential-shaped synapses



(b) LIF sampling with alpha-shaped synapses

Figure 4.17.: Examples for the hidden distributions in the 4 bar example with $W_{\text{exc}} = 3.6 = -W_{\text{inh}}$. Depicted are the results for LIF sampling with CUBA exponential and alpha-shaped synapses. The corresponding COBA results are similar and are therefore omitted.

D_{KL} is a trustworthy measure because all contributions to it go in the same direction and are thus added up. In contrast to alpha-shaped LIF sampling, we see that exponential-shaped LIF sampling just marginally improves mixing compared to Gibbs. AST manages again to reproduce the theoretical distribution over the whole range of weights.
These results demonstrate that the example we discussed in Section 4.2.2 was no special case. If the patterns reach a certain strength then we obtain systems where mixing is important. And in this situation we saw the same behavior as for large system sizes in Section 4.2.3 Gibbs and exponential-shaped LIF sampling stop mixing and usually stay in just one mode. Alpha-shaped LIF sampling is still able to mix between the modes on the cost of producing additional states while AST reproduces the distribution in every case very well.

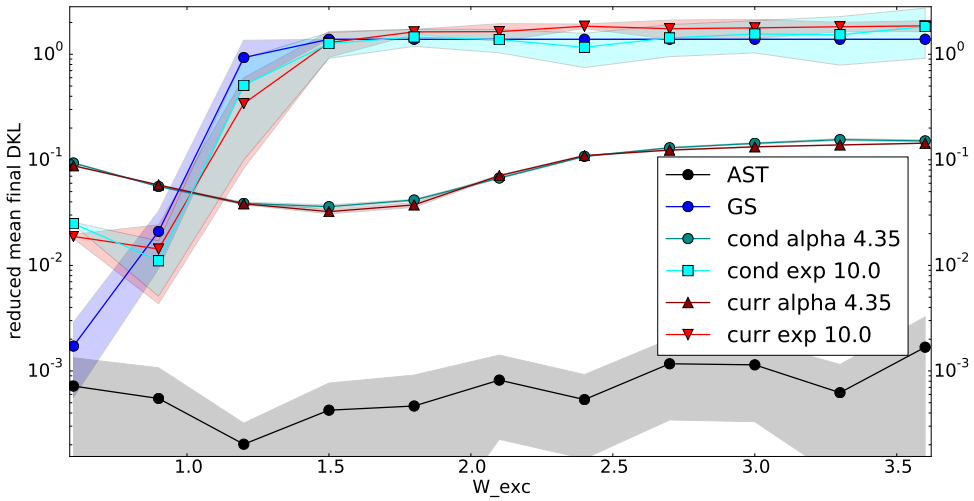


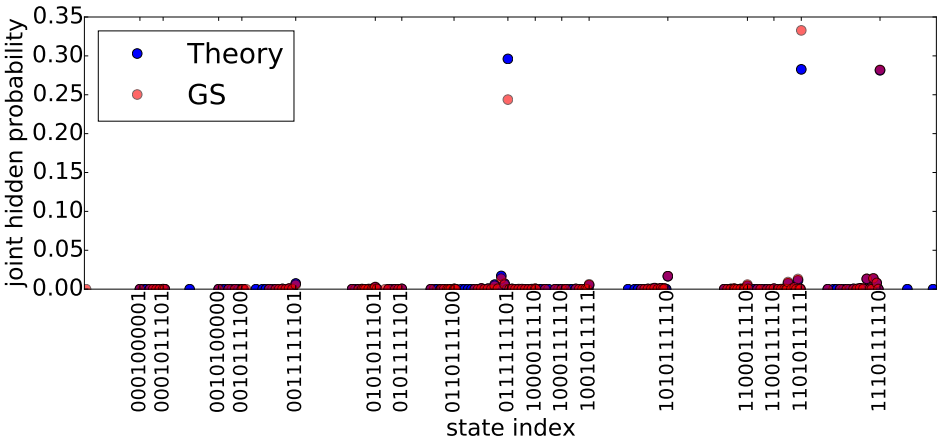
Figure 4.18.: Mean final reduced D_{KL} values corresponding to Fig. 4.16.

4.3. MNIST Visualizations

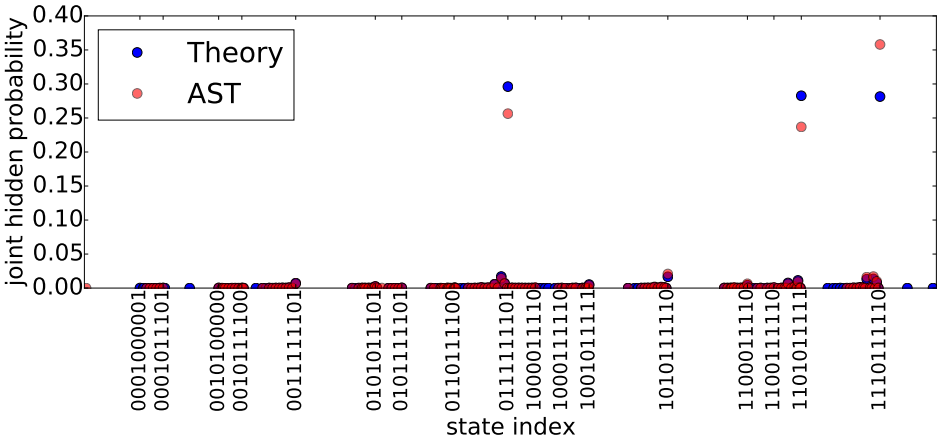
In this section we apply the same sampling algorithms as before to generate samples in RBMs which we have trained on pictures of the MNIST data set of handwritten digits (*LeCun and Cortes*, 1998). We will start with a small system where we trained an RBM on just three digits. This has the advantage that we can apply the same methods as in the previous sections to investigate the behavior of the system. Additionally, we will apply the visualization techniques presented in Section 3.10 to check whether they are in accordance to the previously used methods.
Afterwards, we consider RBMs trained on all ten digits. Here, due to the larger size of the systems, we can only apply the visualization techniques from Section 3.10.

4.3.1. MNIST 3 Digits

In this section we consider an RBM trained on three MNIST digits. One "0", one "3" and one "4". To accelerate the calculation we reduced the dimension of the three digits from 28^2 which is the standard for MNIST to 12^2. Hence we use an RBM with 12^2 visible units and 10 hidden units. The number of hidden units is chosen such that we obtain a large enough system to sufficiently learn the digits, but which is at the same time small enough to allow an evaluation of the theoretical hidden distribution. For the following results we trained the RBM with PCD (see Section 3.9.2). The used learning parameter are listed in Table A.4. We made an effort to stop the learning at a point where the 3 digits are imprinted onto the network equally strong, which could be controlled by observing the hidden distribution.
As in the previous sections, we compare 10^6 ms of LIF sampling with 10^5 Gibbs/AST sampling steps. In Fig. 4.19 we see the results for the hidden distributions for Gibbs and AST. In Fig. 4.21b one can look up which digits are represented by the most probable hidden states. Using this, we can see that the three dominant states in the theoretical distribution indeed represent the three different digits. Both Gibbs sampling and AST seem to reproduce the theoretical distribution and are therefore able to mix between the three modes. In Fig. 4.20 the corresponding hidden distributions for LIF sampling are shown. As there is no significant difference in the results of the CUBA and COBA models we will just regard the results for exponential and alpha-shaped CUBA LIF sampling.
They show that exponential-shaped LIF sampling has difficulties to mix between the modes, even more than Gibbs sampling. Alpha-shaped LIF sampling, however, is able to reproduce the three dominant modes but with larger deviations compared to AST.
These results demonstrate how well the sampler can represent the hidden layer. It is very likely, but not guaranteed, that the visible layer will follow this behavior. We will therefore use the visualization techniques from Section 3.10 to investigate this further.

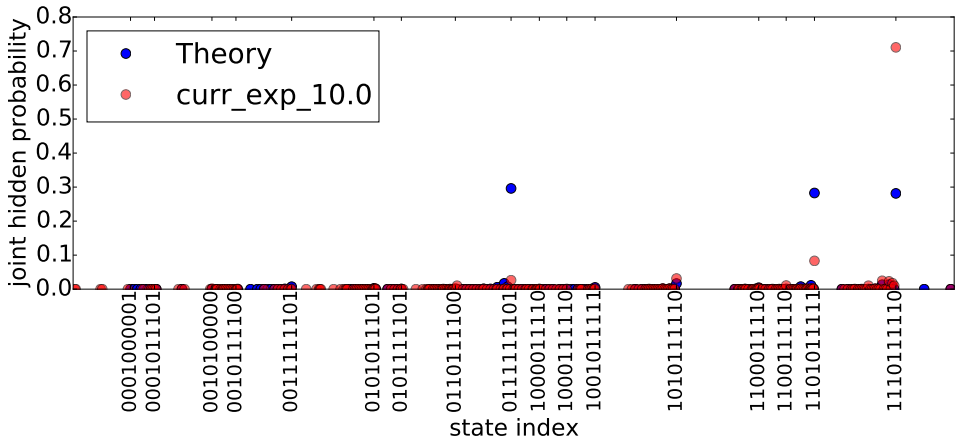


(a) Gibbs sampling

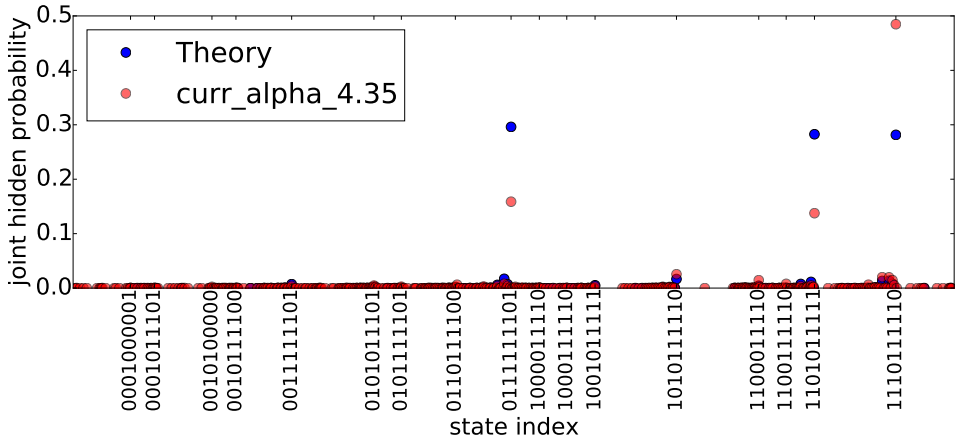


(b) AST

Figure 4.19.: Theoretical hidden distribution compared to the sample distribution for Gibbs sampling and AST. The possible states of the 10 hidden units are represented by the x-axis, while the y-axis shows the probability of the states. The binary strings which label the states show which of the 10 hidden units are activated (1) or deactivated (0). As it is impossible to label all 2^{10} states, we just label the states explicitly which have one of the 100 largest probabilities and which do not overlap with other labels. The RBM has been trained on three MNIST digits. The three highest dots represent the states of the three digits as can be seen in Fig. 4.21.

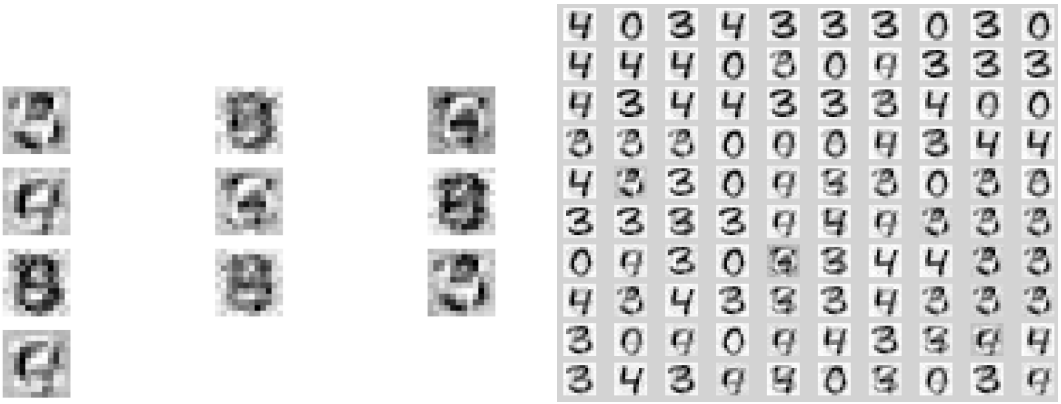


(a) Exponential-shaped LIF sampling



(b) Alpha-shaped LIF sampling

Figure 4.20.: Theoretical hidden distribution compared to the sampled distribution for CUBA alpha and exponential-shaped LIF sampling. The setting is the same as in Fig. 4.19. The corresponding results for COBA LIF sampling are similar and therefore omitted.



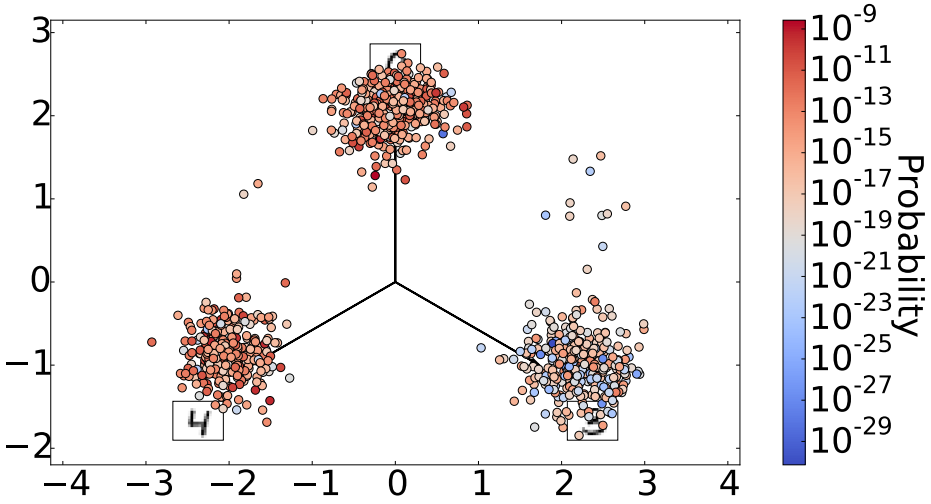
(a) Receptive fields of the hidden units

(b) Receptive fields of the 100 most probable hidden states

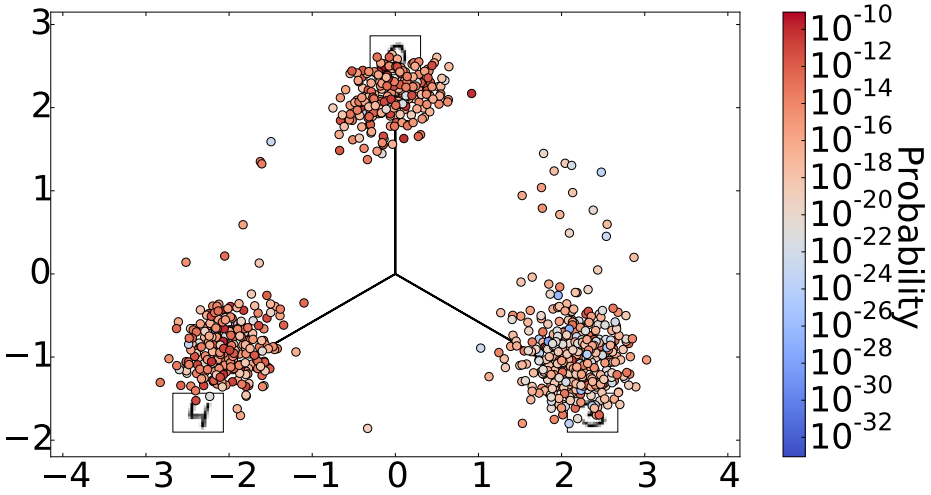
Figure 4.21.: In Fig. 4.21a the receptive fields (Section 3.2.1) of the hidden units after training on the three MNIST digits are shown. They show that the hidden units indeed represent certain features in the input data set, as most of them look like a mixture between the three digits.
In Fig. 4.21b we determined the "receptive fields" of the 100 hidden states with the largest probabilities (shown e.g. in Fig. 4.19). To determine them, for each state the mean receptive fields of the active hidden units were calculated.

We will first look at the results for the star plots (Section 3.10.1). We see that they confirm our assumptions from the hidden distributions. For the abstract samplers in Fig. 4.22 we see that both are able to produce samples which are centered around the directions representing the three digits. Furthermore, as the sample clusters for the 3 digits have the same size, the sampler seem to mix well between the modes. Also the probability of the sampled states is relatively high in most cases. Exponential-shaped LIF sampling (Fig. 4.23a) exhibits much more samples from the "3" mode than from the others. This shows that it has problems to traverse between the modes. Alpha-shaped LIF sampling (Fig. 4.23b), however, seems to be able to mix into all modes but it produces slightly more samples from the "3" mode as well. It also samples from more intermediate states with lower probability than the other sampling algorithms.
In Fig. 4.24 and Fig. 4.25 we see the corresponding results for PCA (Section 3.10.2). One can see that PCA seems to be sufficient for this example to produce good visualization results. It is able too separate the samples very well for all samplers. The results for the sampling algorithms itself correspond to our prior results, but the connections between consecutive samples reveals a new insight into the mixing behavior of the algorithms. It shows that there are much less connections between the clusters for Gibbs sampling than for AST. This demonstrates that, although Gibbs sampling reproduces the theoretical distribution, it is harder for it to mix between the modes. Considering the corresponding results for LIF sampling reveals that exponential-shaped LIF sampling has indeed large

difficulties to mix between the three modes. Alpha-shaped LIF sampling mixes similarly to AST, however, one can see that it generates more "3"s than other digits, as exhibited by the hidden distribution in Fig. 4.20b. Furthermore, one can see the intermediate states which were also visible in the star plot (Fig. 4.23b). Here we see that these samples look like mixtures between the digits of the neighboring clusters. This explains their low probability in the star plot, because the receptive fields of the likely hidden states (Fig. 4.21b) show that just clearly visible states appear with a high probability in the hidden distribution.
The same results with the t-SNE method (Section 3.10.4) in Fig. 4.26 and Fig. 4.27 show a small improvement of the visualization compared to PCA. The three clusters are less compact than in PCA but still separated very well. This makes it possible to see additional structures within the generated samples of one cluster. Furthermore, it makes the connections between the clusters better visible, because the lines are less focused on just small areas. In summary, the visualization results for the star plots, PCA and t-SNE work well for this example. Their results are consistent with each other as well as with the results from the hidden distributions. This also confirms that the behavior of the visible layer follows the hidden layer as we have also seen for the bar example before (Section 4.2.2). The results tell us that in this example mixing is just a small issue. Gibbs sampling mixed slower than AST, but it could still reproduce the theoretical distribution. LIF sampling with exponential-shaped synapses showed a worse mixing performance than Gibbs sampling. It was the only sampling algorithm which hardly mixed in this example and therefore produced an imbalanced sample distribution. LIF sampling with alpha-shaped synapses had, as we have seen before, no problems with mixing. However, it tended to yield slightly imbalanced distributions as well and exhibited some intermediate samples.

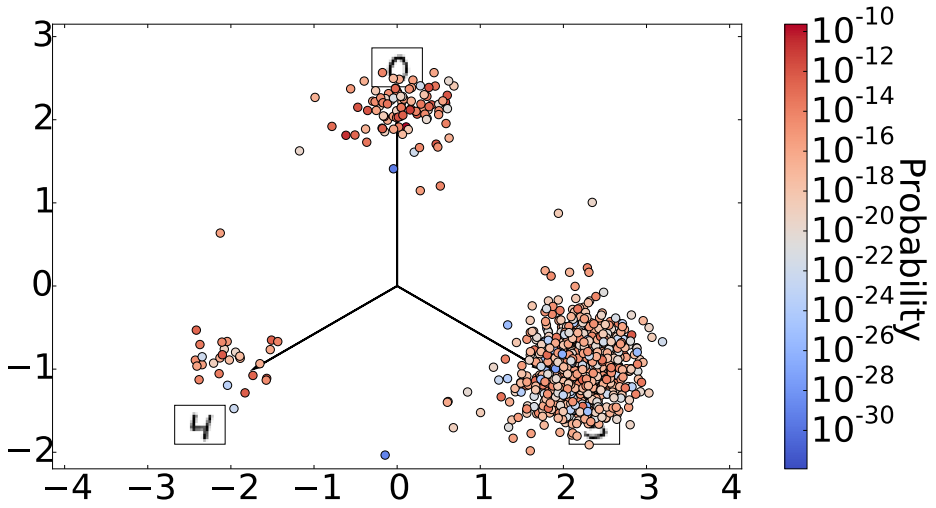


(a) Gibbs sampling

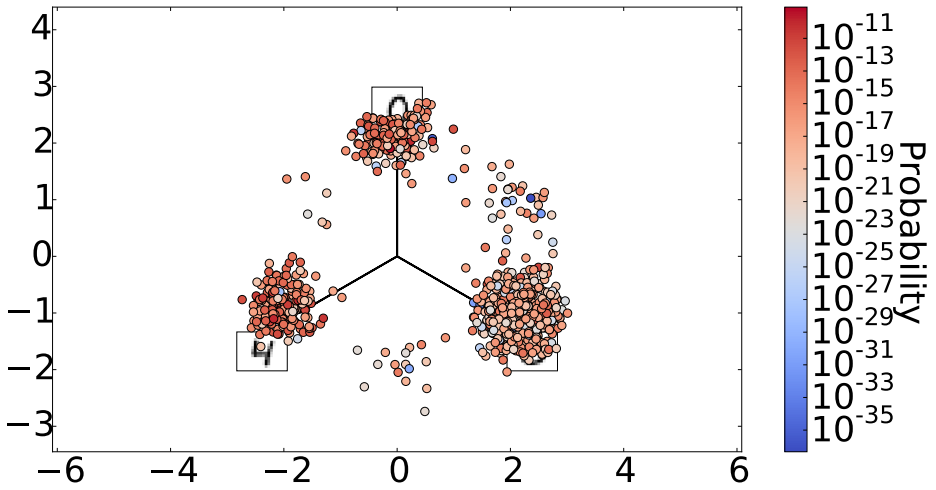


(b) AST

Figure 4.22.: Star plots for Gibbs sampling and AST. We use the method described in Section 3.10.1 to compute the two-dimensional positions from the visible part of the samples. The three pictures of a "0", "3" and a "4" denote the axes which represent them. To obtain a clear plot we just showed every 200^{th} sample, resulting in 500 samples in total. The colors correspond to the probabilities of the samples to occur. The typically low probability values for the single samples are due to the large number of possible states (2^{154}).

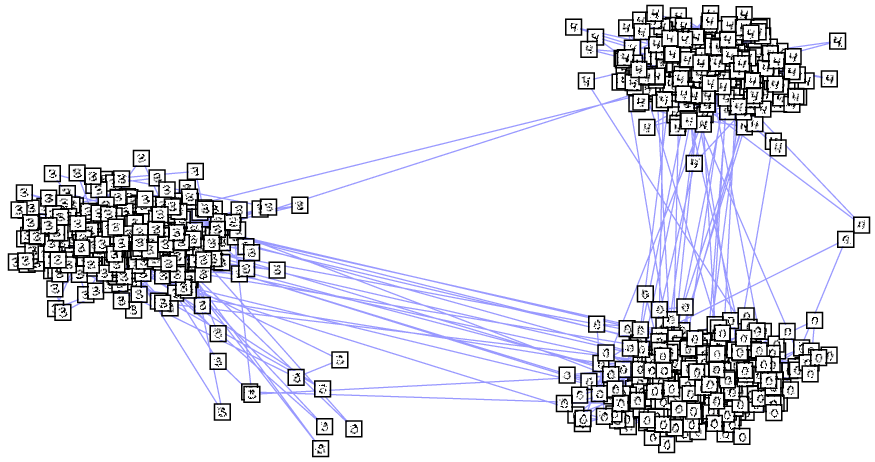


(a) Exponential-shaped LIF sampling

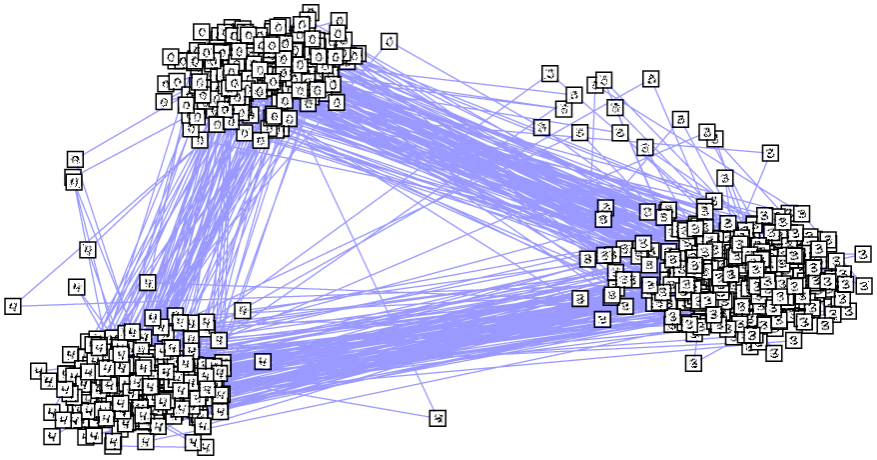


(b) Alpha-shaped LIF sampling

Figure 4.23.: Star plots for CUBA exponential and alpha-shaped LIF sampling. For a detailed description see the corresponding plot for the abstract sampler (Fig. 4.22). Here we show a sample every 2000 ms to obtain 500 samples in total.

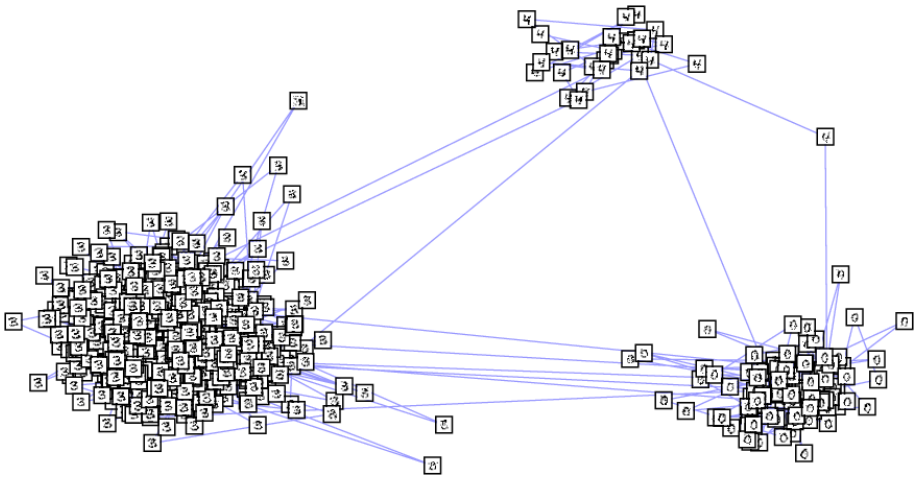


(a) Gibbs sampling

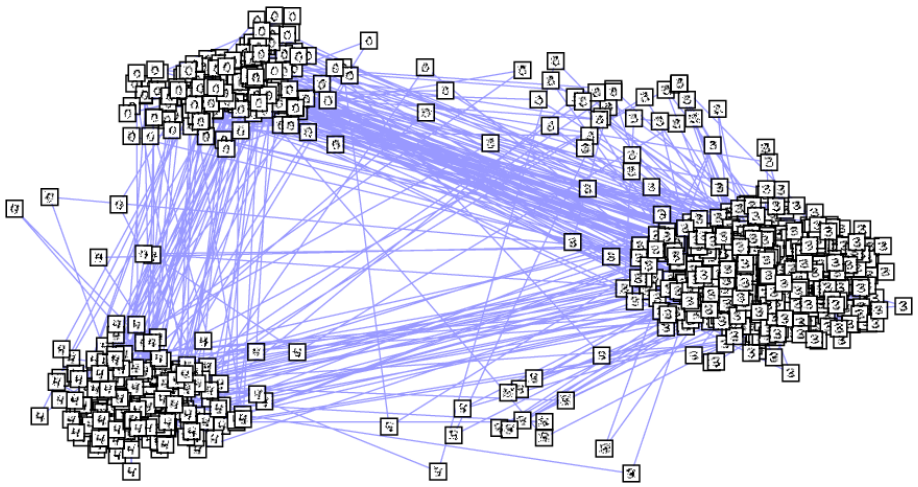


(b) AST

Figure 4.24.: Result of PCA for Gibbs sampling and AST. We omitted the coordinate axes because they don't add any useful information. We used images of the visible samples instead of a normal scatter plot because otherwise it would be not clear which cluster belongs to which digit. This is additionally useful to examine the quality of the generated samples and the correctness of the separation. The samples are the same as for the star plots. The blue lines connect consecutive samples. They give a qualitative impression of how well the sampler mixes between the clusters.

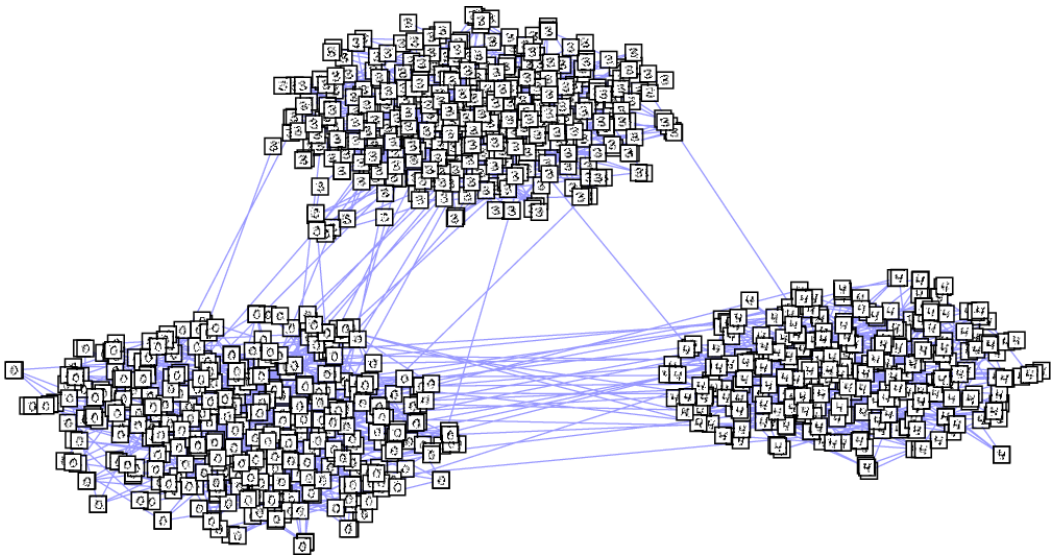


(a) Exponential-shaped LIF sampling

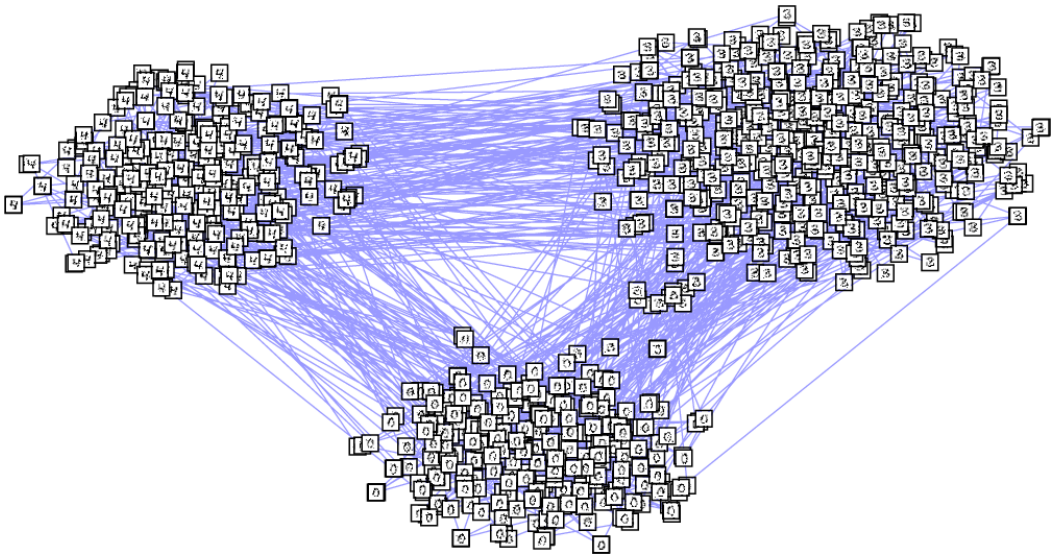


(b) Alpha-shaped LIF sampling

Figure 4.25.: PCA results for CUBA exponential and alpha-shaped LIF sampling. See Fig. 4.24 for a detailed description.

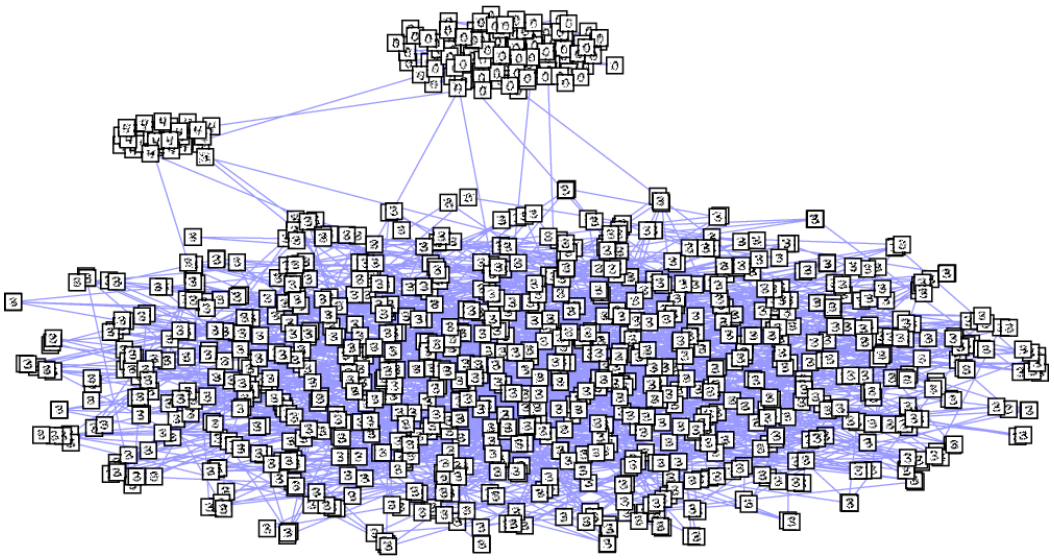


(a) Gibbs sampling

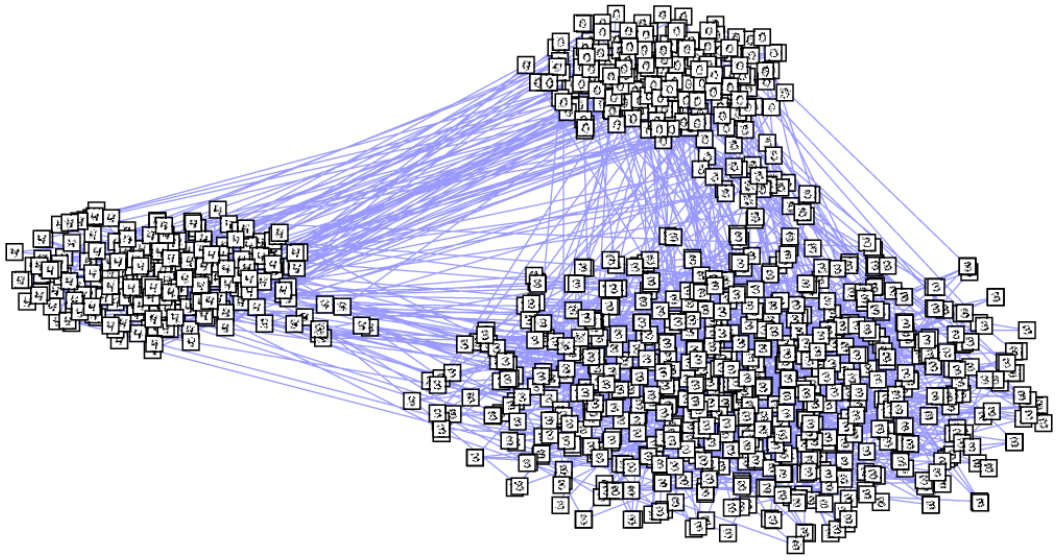


(b) AST

Figure 4.26.: Result of the t-SNE visualization for Gibbs sampling and AST. The plotting method is identical to Fig. 4.24. The used t-SNE parameters are listed in Table A.9.



(a) Exponential-shaped LIF sampling



(b) Alpha-shaped LIF sampling

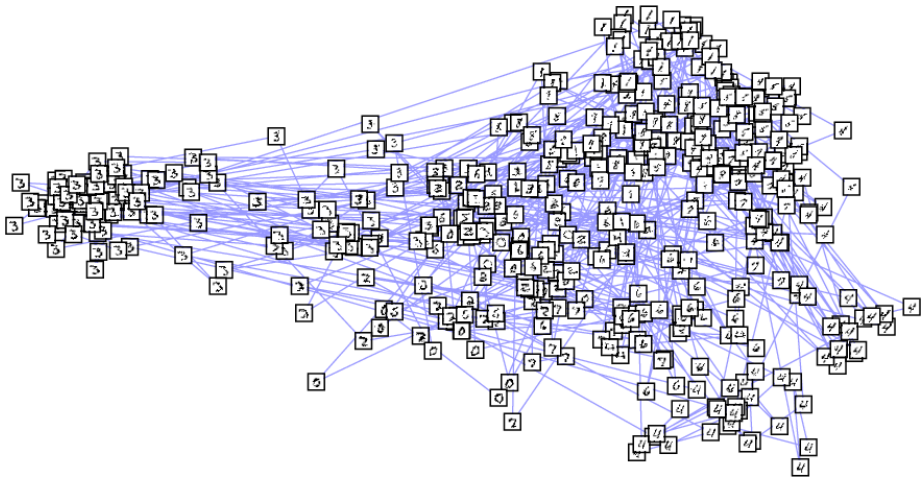
Figure 4.27.: Result of the t-SNE visualization for CUBA exponential and alpha-shaped LIF sampling. The plotting method is identical to Fig. 4.24. The used t-SNE parameters are listed in Table A.9.

4.3.2. MNIST 10 Digits

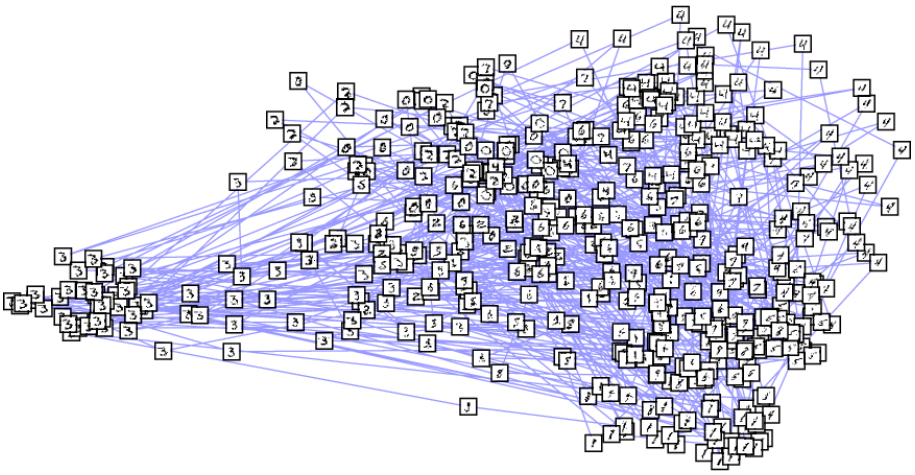
In this section we consider the sampling algorithms for all ten digits of the MNIST data set. However, to keep the system size and therefore the learning and sampling time small, we reduce the data set to just 10 images per digit, leading in total to 100 training images. To sufficiently learn this data set we use RBMs with 100 hidden units. As a consequence, it is not possible anymore to evaluate the hidden distribution. Furthermore, the star plots are not suited to be used for more than 3 digits, as mentioned in Section 3.10.1. In this section we are therefore restricted to use only PCA and t-SNE for data visualization.

Reduced Image Size

To facilitate learning and sampling we use in this part MNIST images which have been reduced to 12x12 pixels, like in the 3 digit example before. Hence, we consider an RBM with 144 visible and 100 hidden units. The parameters we used to train the RBM on this data set are listed in Table A.5. The parameters for sampling are the same as in the 3 digit example. The only difference is that this time we sample for just $5 \cdot 10^5$ ms ($5 \cdot 10^4$ steps). As before, we will show 500 samples in each plot which are equally distributed over the sampling duration. The results for PCA are shown in Fig. 4.28 for Gibbs and AST and Fig. 4.29 for LIF sampling.
They show that all sampling algorithms generate well recognizable samples, which demonstrates that learning and the state generation works well for this system. The PCA results are, however, difficult to read. Usually just one or two digit classes are separated very well, while the rest is still crowded together.
In Fig. 4.30 and Fig. 4.31 we will therefore consider the corresponding results for t-SNE, which is designed to solve this problem. One can see that these plots are indeed much easier to interpret. The sampled states are better grouped into digit classes in all four cases. In some few cases only samples from similar digit classes or unclear samples are mixed together. The better grouping makes it also easier to qualitatively judge the connections between the digits. From this we can see that the sampling algorithms show the same behavior as in the 3 digit example before. Gibbs sampling (Fig. 4.30a) can mix between the modes, but it has sparser connections between the different classes when compared to AST. AST (Fig. 4.30b) has no mixing issues and reproduces the data set very well. In case of exponential-shaped LIF sampling (Fig. 4.31a) we again see that it is the only method with strong mixing issues. It could not reproduce every digit class and it has just sparse connections as well. Instead, it stayed in the zero mode most of the time. Alpha-shaped LIF sampling (Fig. 4.31b) had no mixing issues, but, as before, it led to less balanced digit clusters and generated more intermediate samples.
The comparison between PCA and t-SNE shows that the extra effort to tune and apply t-SNE is justified for this 10 digit example, because its results are easier to interpret. We learn from the results that mixing seems to be no big issue except for exponential-shaped LIF sampling, even in this large system. The reason is that the digits typically largely overlap with each other, which simplifies the transition between the modes.

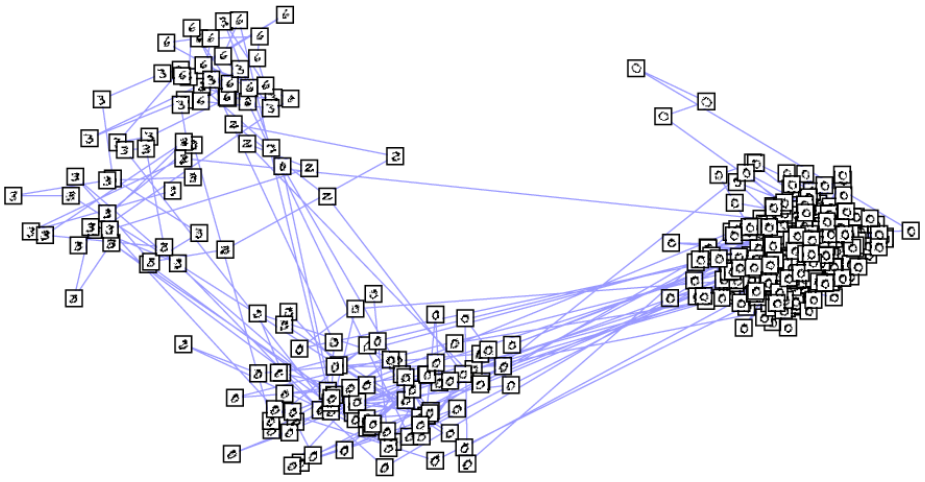


(a) Gibbs sampling

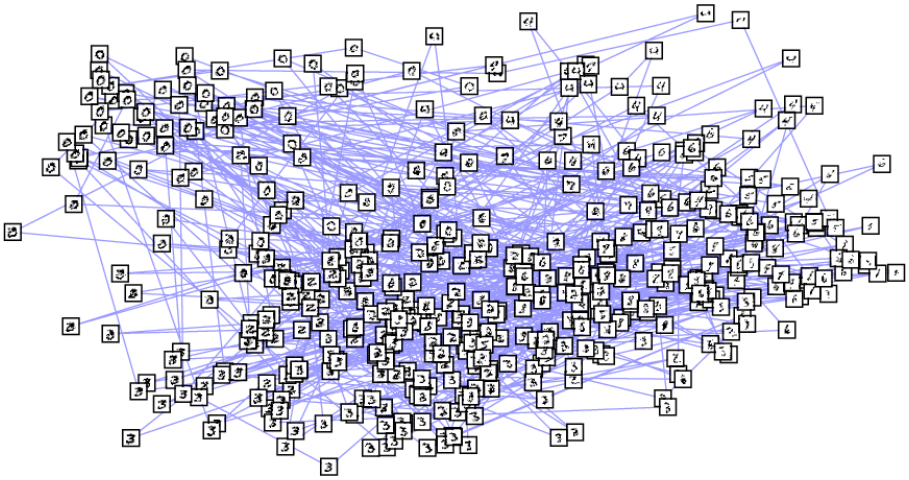


(b) AST

Figure 4.28.: PCA results for for Gibbs sampling and AST. See Fig. 4.24 for a detailed description of the plotting details.

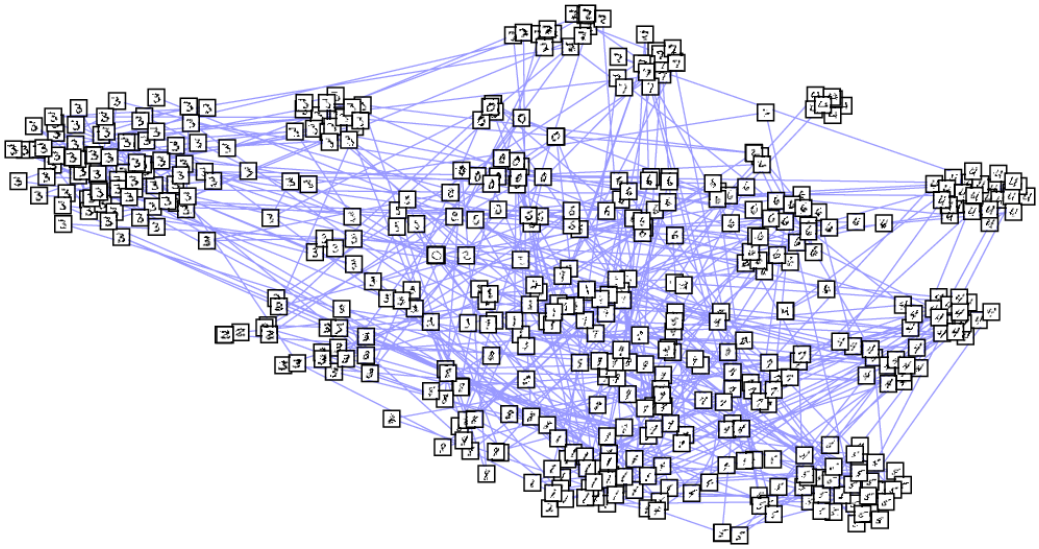


(a) Exponential-shaped LIF sampling

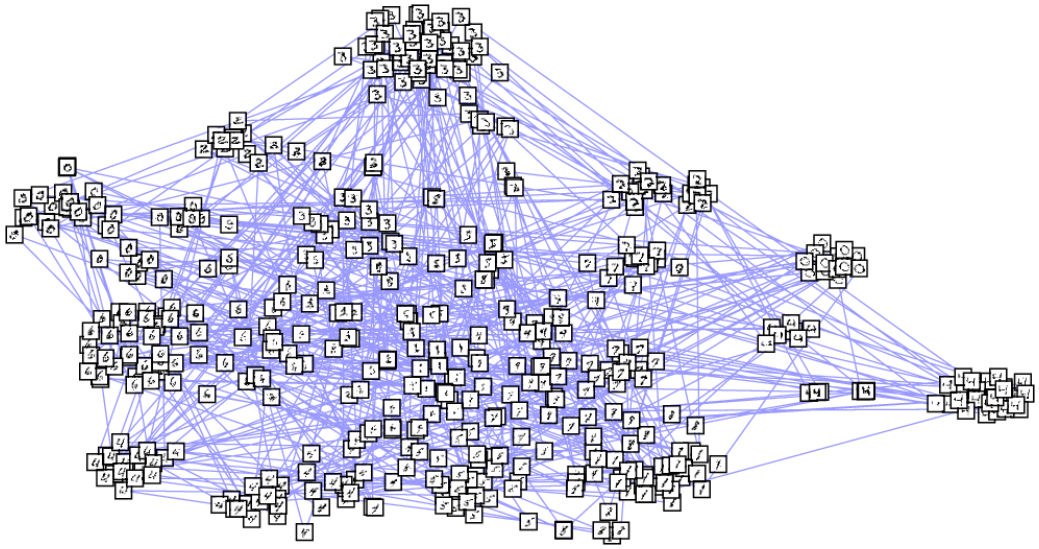


(b) Alpha-shaped LIF sampling

Figure 4.29.: PCA results for CUBA exponential and alpha-shaped LIF sampling. The corresponding COBA results are similar and therefore omitted. See Fig. 4.24 for a detailed description of the plotting details.

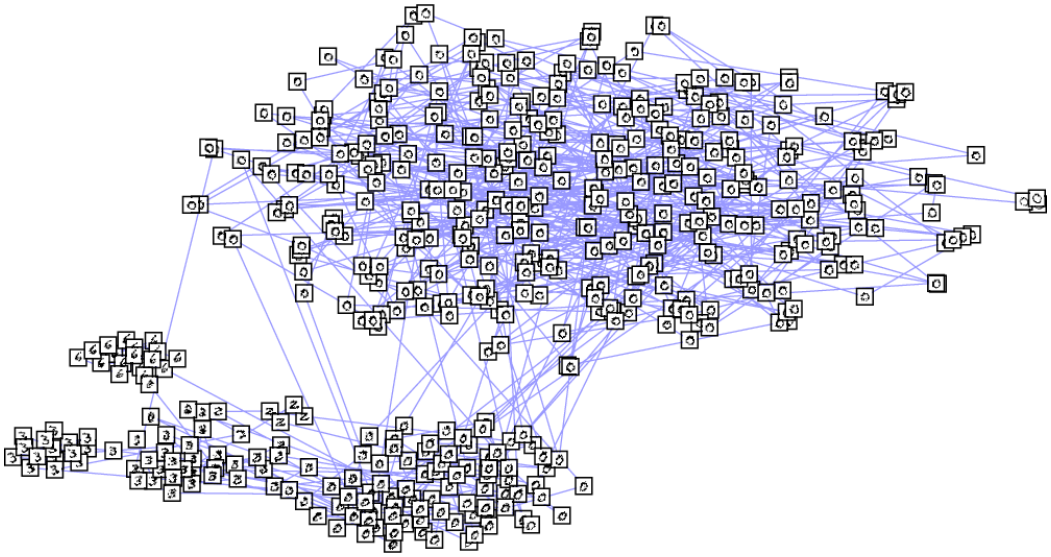


(a) Gibbs sampling

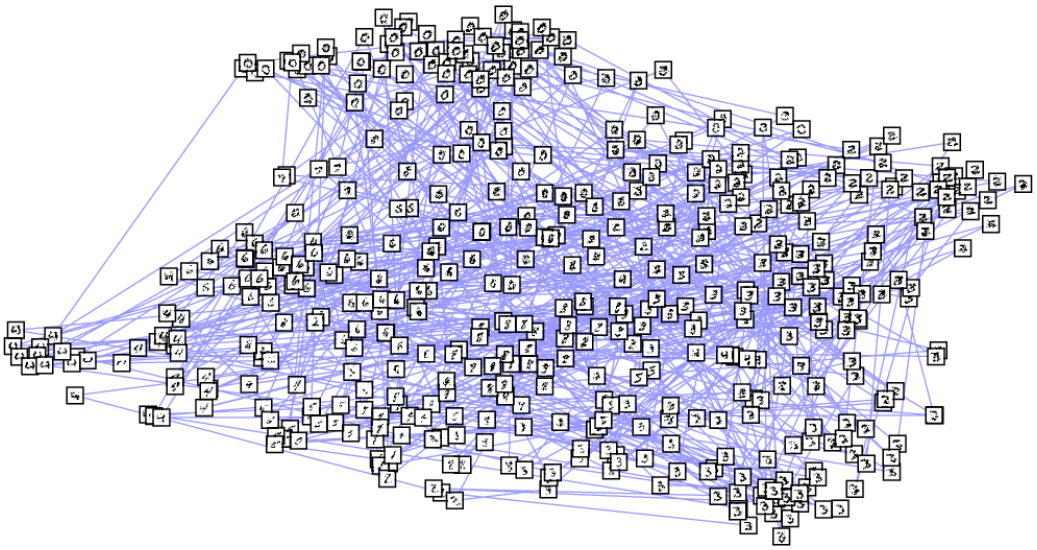


(b) AST

Figure 4.30.: t-SNE results for for Gibbs sampling and AST. See Fig. 4.24 for a detailed description of the plotting details. The used t-SNE parameters are listed in Table A.9.



(a) Exponential-shaped LIF sampling

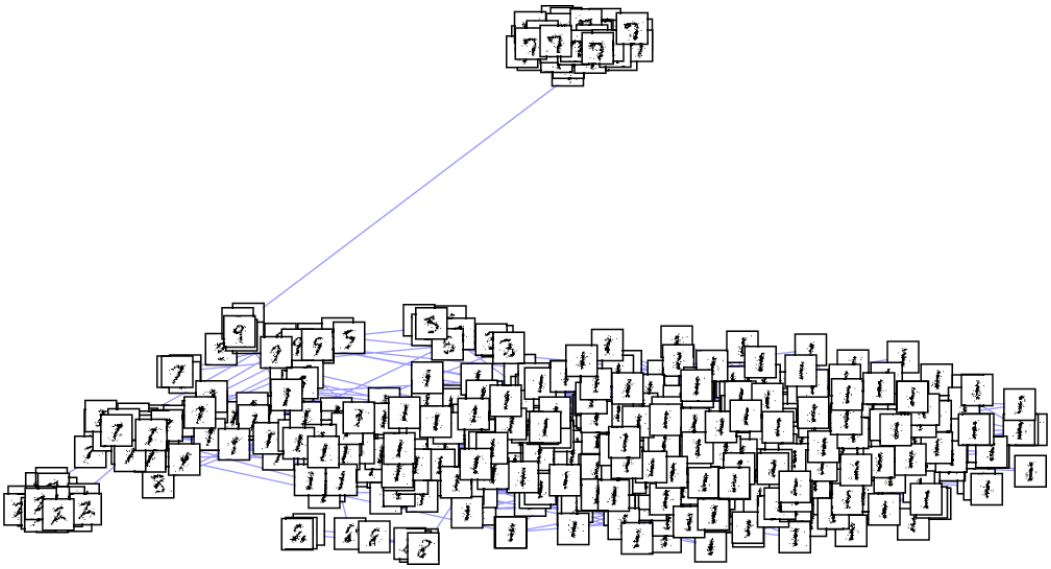


(b) Alpha-shaped LIF sampling

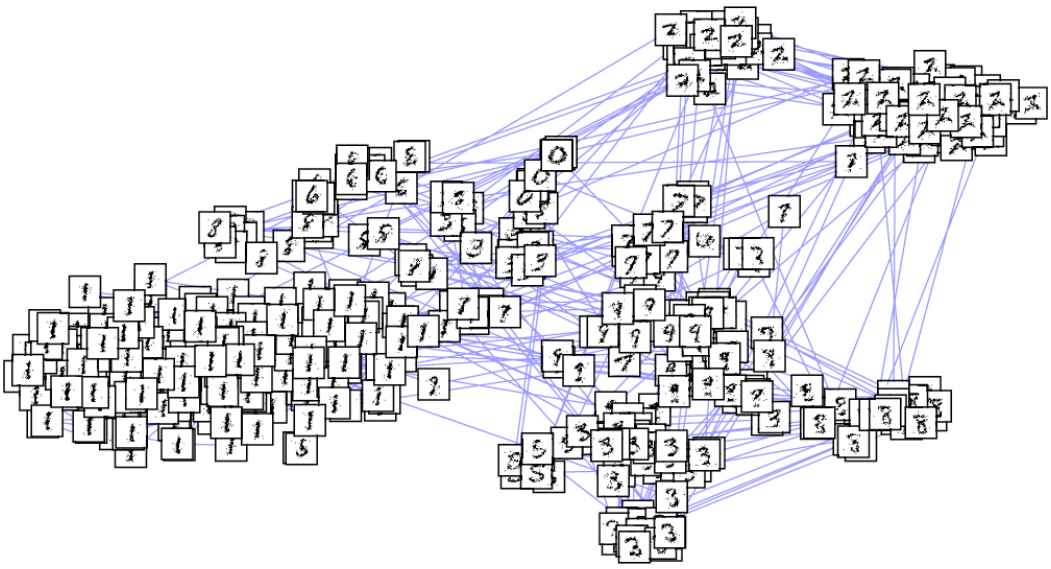
Figure 4.31.: t-SNE results for CUBA exponential and alpha-shaped LIF sampling. The corresponding COBA results are similar and therefore omitted. See Fig. 4.24 for a detailed description of the plotting details. The used t-SNE parameters are listed in Table A.9.

Normal Image Size

In this section, we consider the same MNIST data set as in the previous section but this time with standard size, namely 28x28 pixels. We therefore consider an RBM with 784 visible and 100 hidden units. The parameters used for training the RBM on the data set are listed in Table A.6. For sampling we use the same parameters as in the previous section. The PCA results lead in this system like in the previous section to results which are difficult to interpret. We will therefore omit them and consider in Fig. 4.32 and Fig. 4.33 directly the t-SNE results for abstract and LIF sampling respectively. They demonstrate that in this regime mixing seems to be a big issue. The Gibbs sampler in Fig. 4.32a barely mixes. It stays merely in the "1" mode. AST in Fig. 4.32b is able to mix but this happens much slower than in the 12x12 pixels case in Fig. 4.30b, which can be seen in the sparser connections between the clusters and the more unbalanced clusters. For LIF sampling, we again see the usual behavior. Exponential-shaped LIF sampling in Fig. 4.33a does hardly mix while alpha-shaped LIF sampling in Fig. 4.33a has much less problems.
Comparing AST and alpha-shaped LIF reveals that the LIF sampler mixes much more often between clusters, but it does not reach every digit class. Ones and sevens are for example missing. Furthermore, as before, alpha-shaped LIF produces more intermediate digits than AST. t-SNE is therefore not able to clearly separate the digits into several distinct clusters. Summed up, this confirms the impressions from the 10 digit example with 12x12 pixels and the 3 digit example. However, here mixing is a much bigger issue. This can be explained with the larger size of the visible layer. In the 28x28 case much more visible units have to change their state to switch a mode than in the 12x12 case, which makes such a transition more unlikely.

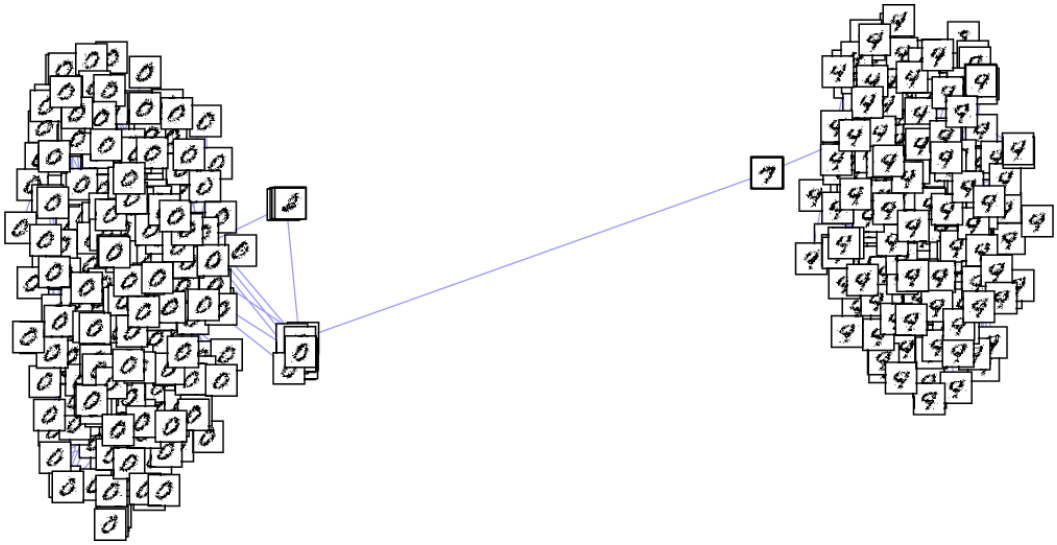


(a) Gibbs sampling

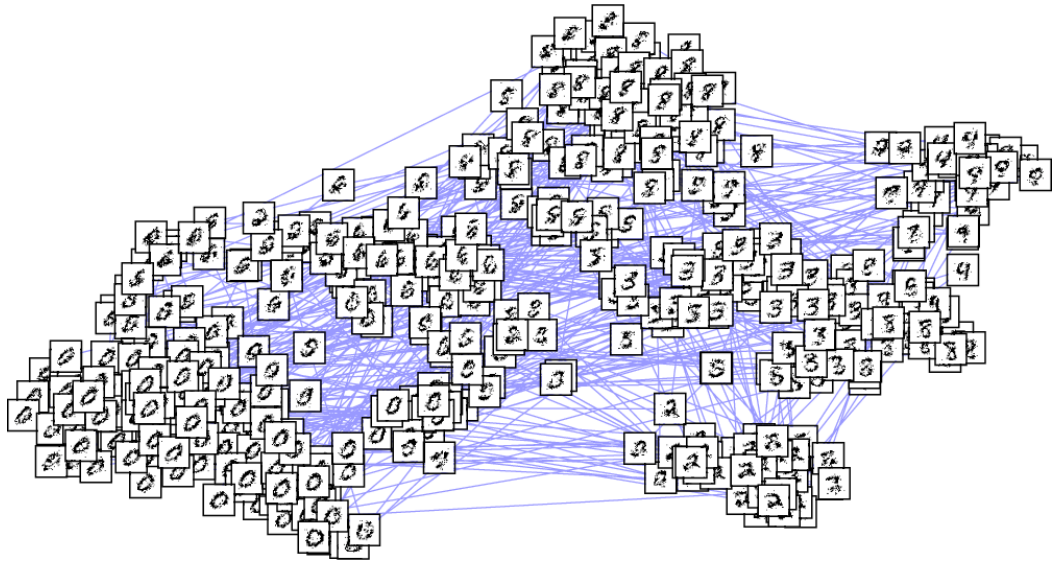


(b) AST

Figure 4.32.: t-SNE results for for Gibbs sampling and AST. See Fig. 4.26 for a detailed description of the plotting details.



(a) Exponential-shaped LIF sampling



(b) Alpha-shaped LIF sampling

Figure 4.33.: t-SNE results for CUBA exponential and alpha-shaped LIF sampling. The corresponding COBA results are similar and therefore omitted. See Fig. 4.26 for a detailed description of the plotting details.

4.4. Discussion

In this chapter we learned that in simple examples, like the homogeneous distributions in Section 4.1.1, all samplers are able to reproduce the theoretical distribution quite well. However, we saw a saturation for LIF sampling when the sampled distribution comes close to the theoretical one, due to the different shape of the post-synaptic potentials (see Fig. 3.7). When considering distributions manually tuned to be difficult in terms of mixing, like in the bar example, we saw that LIF sampling with alpha-shaped synapses exhibits a better mixing property than Gibbs sampling. However, LIF with exponential-shaped synapses performs similarly to Gibbs sampling. For the larger MNIST systems, we saw that LIF sampling with exponential-shaped synapses mixed even worse than Gibbs sampling while alpha-shaped LIF still shows better mixing. The reason for this behavior lies in the different PSP shape as well. The discrepancy between the two LIF samplers demonstrates that the exact shape of the PSP appears to be critical. At first glance the deviations of the two LIF sampling methods look similar, but, considering the results, one demonstrates worse mixing while the other one's ability to mix improves. An explanation why the deviations lead to an improved mixing could be a combination of the large overshoot and the low potential level at the end of the refractory time in Fig. 3.7. This leads to a large variation of the potential during the refractory period compared to the rectangular shape of the abstract sampler, which can activate neurons in situations where they should in theory not be activated or can prevent them from being activated. This drives the system to explore a larger state space. This effect is present for both synapse models, but it is stronger for alpha-shaped synapses because their variation is larger.
The remaining potential level (*tail*) after the refractory period is, on the other hand, an effect which should reduce the ability to mix. During spike bursts it has no effect because the TSO mechanism compensates it. However, when just single spikes occur or a spike burst is interrupted, the tail acts like a memory of the previous state biasing the neuron to not change its behavior after the refractory period. This effect is smaller for the alpha-shaped model.
Hence, we have a trade-off between two effects. The variation increasing the mixing ability and the tail reducing it. While for the exponential-shaped model the tail seems to outweigh the variation such that it mixes worse than the abstract rectangular shape, it is the opposite for the alpha-shaped model.
This can also explain the appearance of the additional states in the bar example (e.g. in Fig. 4.10b). They appear due to the variation, which also explains why the additional states were more frequent for the alpha-shaped model than for the exponential-shaped one. The intermediate states which appeared for alpha-shaped LIF sampling in the MNIST examples are due to the reason that this mixing mechanism just gradually moves the sampler from one mode to another.

During this chapter we furthermore showed that, as one might intuitively guess, mixing becomes just important for large systems with strongly imprinted patterns, which became especially obvious for the bar pattern (Section 4.2.2). Also, in the MNIST example,

we saw that just after the transition to 28x28 pixels it is easy to find systems where mixing becomes important. The examples with random distributions showed, on the other hand, that large systems alone, even with large weights, are not sufficient to make mixing important, as they lack patterns. The reason for this is that, in order to make mixing crucial, we need a probability landscape similar to Fig. 3.2a. This means we need several regions in the state space which have a high probability and are separated from each other by regions of low probability. Both appear in systems with strongly imprinted patterns. With increasing system size the state space grows exponentially. This makes the distances between high-probability regions larger. As a consequence, traversing these distances becomes more unlikely, as many units have to change their value, to reach another mode with high probability.

For the visualization techniques we saw that the hidden distributions, the star plots, PCA and t-SNE delivered consistent results which give a good impression of mixing. The hidden distributions are, however, limited to systems where the hidden layer is small enough. The star plot is limited to the 3 digit example and PCA delivers also only good results if the system is simple. t-SNE is the only technique which delivered good results for large and complicated systems like MNIST with 10 digits. It is difficult to adjust, though, as it requires a lot of fine tuning of parameters. Yet, we could take advantage of the experience of the authors of *Van der Maaten and Hinton* (2008) because they described their parameter choices for the MNIST data set. However, applying this method to other examples like the bar patterns would require a new fine tuning.
The D_{KL} evaluation has the advantage that it is the only technique tested in this thesis which delivers quantitative results. It can therefore also measure the mean behavior for several runs of the sampling algorithms. Yet, it is difficult to handle for inhomogeneous distributions because it is very sensitive to the appearance of additional sampled states, whose theoretical values are nearly zero, as seen in the bar example in Section 4.2.2. Considering a reduced D_{KL} like in Section 4.2.2 has, on the other hand, the disadvantage that it is not bound to be positive anymore, which makes it difficult to be interpreted. Furthermore, D_{KL} evaluations only work for small systems or RBMs with a small hidden or visible layer.

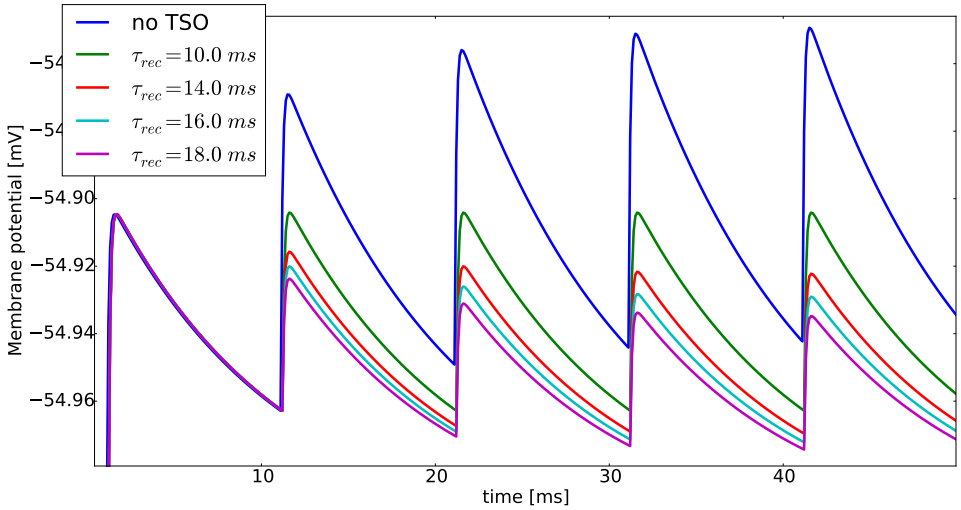
5. Using Short-Term Plasticity to Improve Mixing

In Chapter 4 we saw that LIF sampling with exponential-shaped synapses had issues to reproduce its theoretical distribution in systems where mixing is important. In these situations it performed even worse than standard Gibbs sampling. However, currently only exponential-shaped synapses are realized on the Spikey or HICANN chip (Section 2.3). As an implementation on these systems is our final goal and the reason why we considered LIF sampling in the first place, it would be highly desirable to find a mechanism which could fix this mixing issue. Serendipitously, we discovered that the short-term plasticity mechanism, discussed in Section 3.7, namely the Tsodyks-Markram model (TSO), can be applied to achieve this. It has furthermore the advantage that it is in a similar form already implemented on HICANN and Spikey.
This chapter begins with a description of our approach to use TSO to enable mixing. Afterwards it is applied for LIF sampling with exponential and alpha-shaped synapses on the cases from the previous chapter where mixing was an important issue. This is the bar example (Section 5.2) and the MNIST example with 10 digits and standard image size (Section 5.3). Finally, in Section 5.4 an interesting side effect is considered where the TSO mixing approach seems to equalize imbalanced distributions.

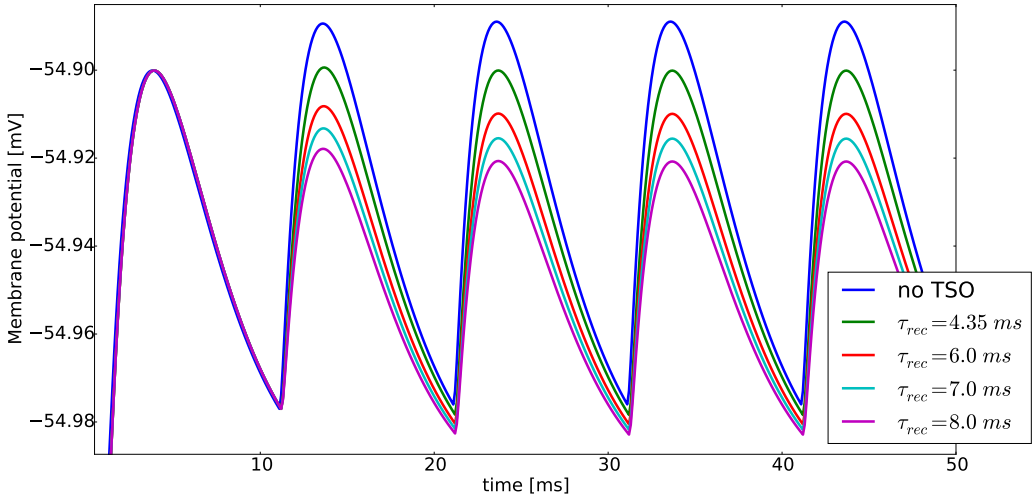
5.1. TSO Mixing Approach

We introduced TSO originally to establish spike bursts with maintained post-synaptic potential PSP heights, as demonstrated in Fig. 3.5. To achieve this, we only used the mechanism of depressing synapses which introduces the recovery time τ_{rec} to slow down the rebuild of synaptic resources. Increasing τ_{rec} beyond the value necessary to maintain the PSP heights leads to a successive decrease of the PSPs for both exponential-shaped and alpha-shaped synapses (Fig. 5.1). This yields to a reduction of the synaptic weight between the pre-synaptic and post-synaptic neuron. As a consequence, when sampling from a Boltzmann distribution, which is determined by its weights and biases, the underlying distribution will be changed. However, the height of the first PSP is always unchanged. Hence, the distribution will just be changed if neurons keep staying in the on-state. In this case the corresponding weights will gradually be reduced which decreases the probability to stay in the same state. Thus, the system is forced to mix when it gets stuck in one state, but, apart from that, it will not be changed. It is important to note that with this mechanism the sampler is not sampling from a Boltzmann distribution anymore, because the weights become asymmetric, as the TSO mechanism affects just the connection from the pre to the post-synaptic neuron. The Boltzmann distribution,

however, is defined with symmetric weights.



(a) Exponential-shaped synapse model



(b) Alpha-shaped synapse model

Figure 5.1.: Simulation of the membrane potential of a neuron which receives a spike every refractory period with refractory time $\tau_{\text{ref}} = 10\,\text{ms}$. Figure 5.1a shows the result for the exponential-shaped synapse model and Fig. 5.1b for the alpha-shaped one. The recovery time constant τ_{rec} of the TSO mechanism is increased, starting from the one which is necessary to establish maintaining PSP heights.

To give an intuitive impression of this mixing principle, we created an image sequence based on the 3 digit MNIST example from Section 4.3.1. It shows the result of LIF sampling with exponential-shaped synapses and TSO with $\tau_{\text{rec}} = 18\,\text{ms}$ to depress the synaptic connections. We extended the star plot visualization to obtain a rough approximation of the temporal evolution of the probability landscape during sampling. The resulting image sequence is shown in Fig. A.3. It shows how every time LIF sampling stays in one mode for a while the probability in the area around this mode is reduced, while the remaining probability landscape is unchanged. This makes it more likely that the sampler leaves the mode. After the sampler switched the mode, one can see how the probability of the previous mode regains its initial value, while the probability of the new mode is reduced.
To compare this with adaptive simulated tempering (AST) (Section 3.3.3) we created a similar image sequence in Fig. A.4 demonstrating the AST mixing principle. We can see how the temperature changes during AST sampling globally change the probability landscape, making it more homogeneous. This is in contrast to the only local changes in case of TSO. While AST is in a high temperature state, the sampler can reach nearly every state. When the temperature is reduced again the current sampling mode can be completely different, which allows mixing. Taking just samples from the ground temperature level ensures that one still reproduces the correct distribution. In case of TSO, however, we are counting every sample as a valid sample. Yet, this has less severe consequences than it would have for AST, because the changes are just locally and more moderate, but it will still change the sampled probability distribution significantly.

5.2. Multimodal Distributions with Artificial Patterns

As a first example to test the above described approach we consider the 4 bar example from Section 4.2.2. Hence, we use an RBM with 144 visible units and 4 hidden units. To make mixing even harder we use this time stronger weights of $W_{\text{exc}} = -W_{\text{inh}} = 2.0$. As in Section 4.2.2, we consider the Kullback-Leibler divergence $D_{KL}(p_{\text{sim}}||p_{\text{theo}})$ between the sampled hidden and the theoretical hidden distribution again. The result for LIF sampling with exponential-shaped synapses and increasing τ_{rec} is shown in Fig. 5.2.

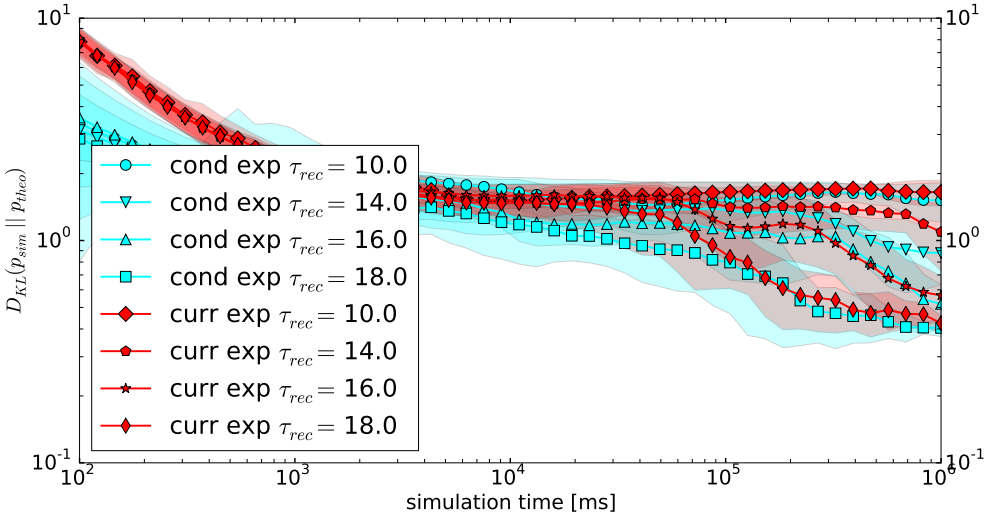
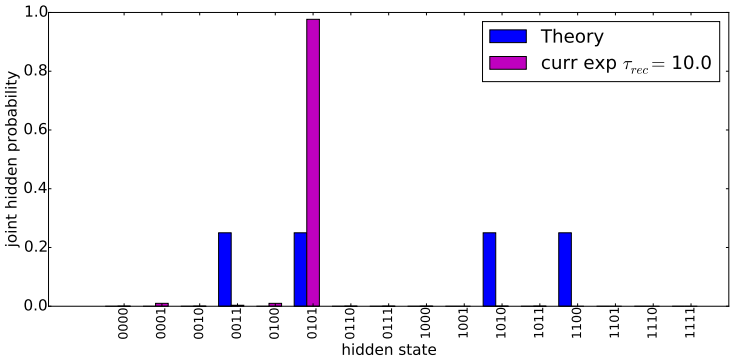
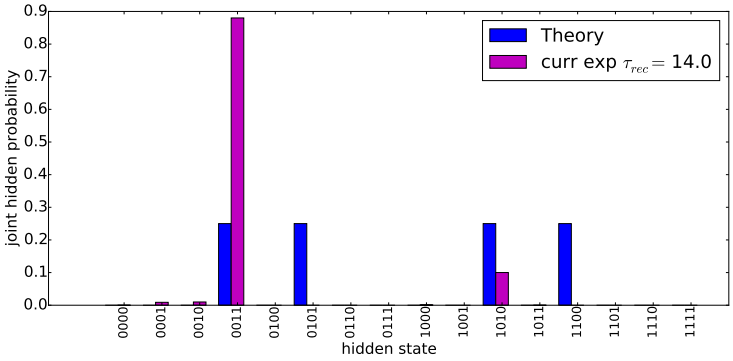


Figure 5.2.: Time evolution of the mean D_{KL} between the sampled hidden and the theoretical hidden distribution for LIF sampling with exponential-shaped synapses. The setup is the bar pattern as in Section 4.2.2 with $W_{\text{exc}} = 2.0$ and $W_{\text{inh}} = -2.0$. The size of the visible layer is 10x10, and the bias $B = -1$ for every unit. We calculate the mean over 10 tries, in which we vary the seed for the random number generator for LIF sampling. The colored area represents the standard deviation. "cond/curr" stand for COBA/CUBA synapses.

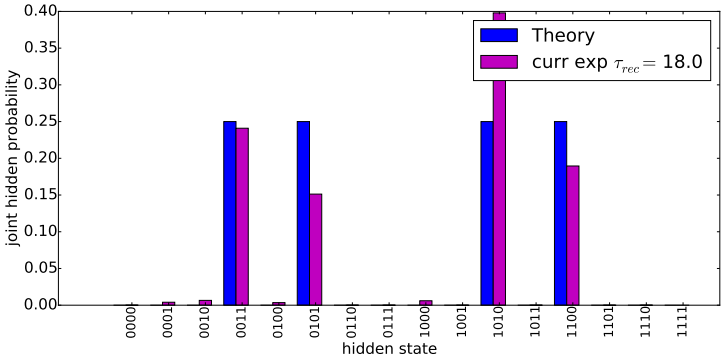
We see that with increasing τ_{rec} the sampling performance is indeed improved, as the D_{KL} between theoretical and sample distribution becomes smaller. Again, as throughout the last chapter, we don't observe a significant difference in the final performance between current-based (CUBA) and conductance-based (COBA) LIF sampling.
To see the reason for the better performance with increasing τ_{rec}, we consider in Fig. 5.3 examples of the hidden distribution for different τ_{rec} values. They show that, as expected, with higher recovery time the sampler is really able to mix between the four modes. This explains why the D_{KL} improves with higher recovery time. We can furthermore see that, as in the corresponding results in Section 4.2.2, the LIF sampler seems to produce a few additional states in all examples, which have a strong impact on the D_{KL} values. They explain why the final D_{KL} values are, despite the improvements, still on a relatively high level. The corresponding behavior of the generated samples is shown in Fig. 5.4. They confirm the impression from the hidden distributions. In case of $\tau_{\text{rec}} = 10\,\text{ms}$, LIF sampling generates nearly only samples of the lower left corner, which corresponds to the [0101] hidden state. For $\tau_{\text{rec}} = 14\,\text{ms}$ LIF sampling still just rarely switches the sampled state but for $\tau_{\text{rec}} = 18\,\text{ms}$ it is able to mix frequently. However, we can also see that by increasing τ_{rec} the sampled patterns become slightly weaker. This is no surprise when we keep in mind that TSO with a high τ_{rec} effectively reduces the weight connections of the active hidden neurons to the visible layer, which weakens the patterns.



(a) Exponential-shaped LIF sampling $\tau_{\text{rec}} = 10\,\text{ms}$

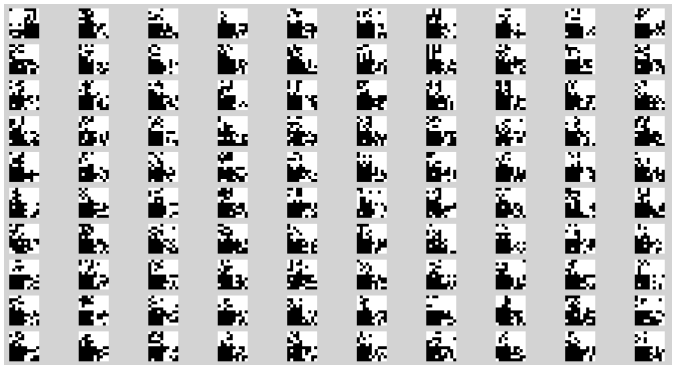


(b) Exponential-shaped LIF sampling $\tau_{\text{rec}} = 14\,\text{ms}$

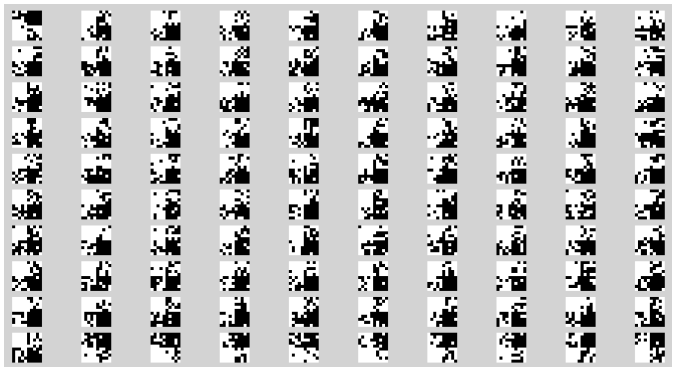


(c) Exponential-shaped LIF sampling $\tau_{\text{rec}} = 18\,\text{ms}$

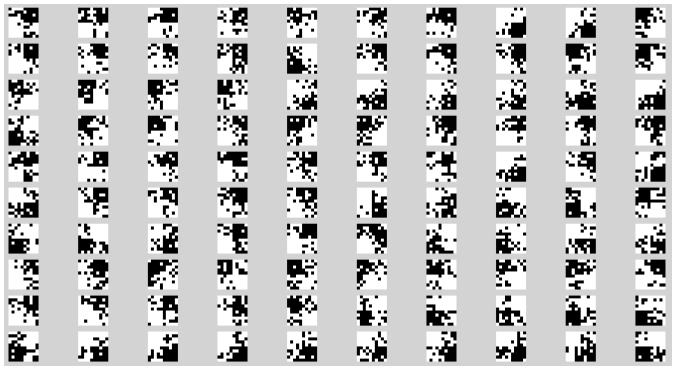
Figure 5.3.: Examples for hidden distributions in the 4 bar example for LIF sampling with CUBA exponential-shaped synapses and increasing recovery time of the TSO mechanism. The final sample distributions of one run in Fig. 5.2 are shown. On the x-axis all 2^4 possible combinations of the hidden layer are listed. The y-axis shows the probabilities for them to appear. The corresponding COBA results exhibit no significant differences and are therefore omitted.



(a) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 10\,\text{ms}$



(b) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 14\,\text{ms}$



(c) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 18\,\text{ms}$

Figure 5.4.: Samples of the visible layer for exponential-shaped LIF sampling with increasing recovery time corresponding to the hidden distributions in Fig. 5.3. The visible units are arranged in 10x10 images as the imprinted bar patterns. The black and white dots denote units which are activated and deactivated respectively. For each sampler 100 samples, taken equally distributed over the sampling time, are shown. The samples should be read from left to right and up to down.

In the following, we will consider the corresponding results for LIF sampling with alpha-shaped synapses. The D_{KL} evolution is shown in Fig. 5.5.

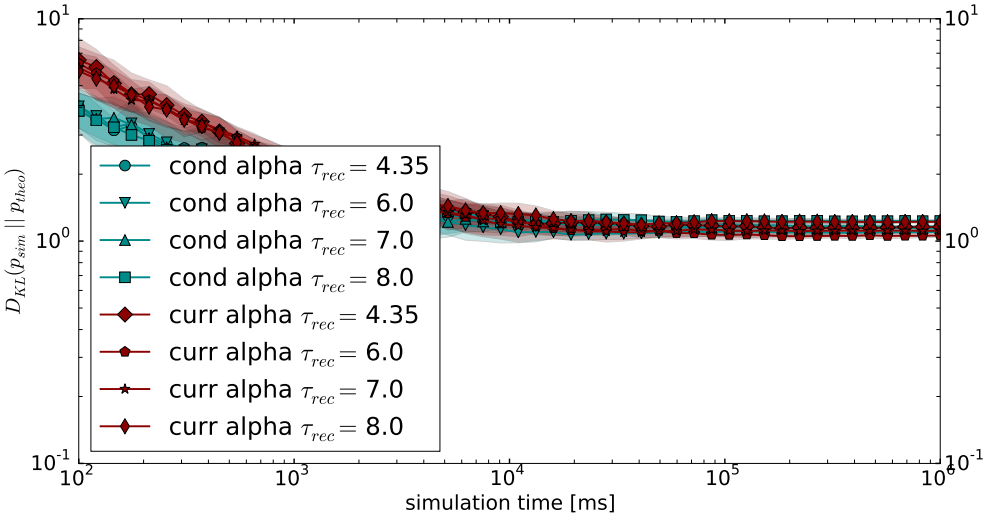
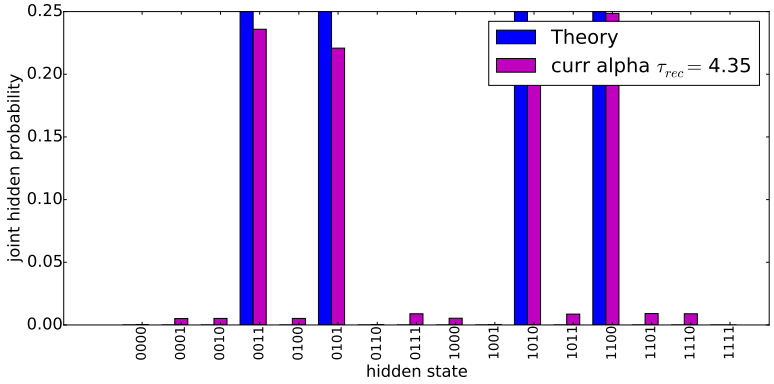
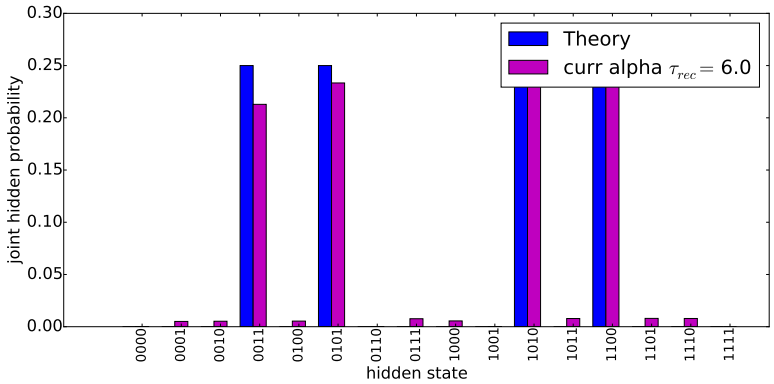


Figure 5.5.: Time evolution of the mean D_{KL} values for LIF sampling with alpha-shaped synapses corresponding to the result for exponential-shaped synapses in Fig. 5.2.

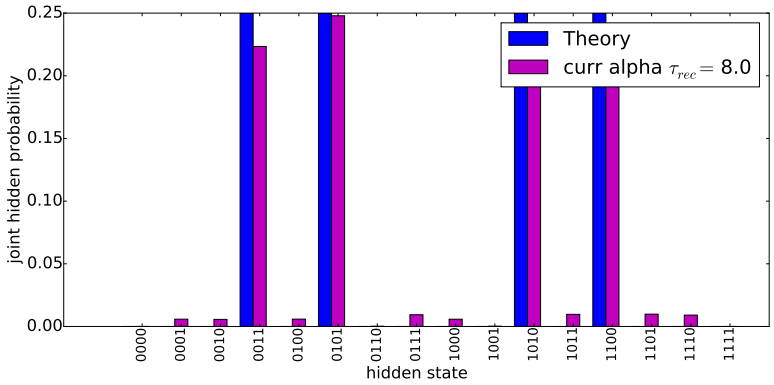
We see that, independent of sampling method and recovery time, the D_{KL} value settles on nearly the same high value. We observed these high D_{KL} values in Section 4.2.2 as well. The reason were additional states appearing in the hidden distributions, which strongly affect the D_{KL} values. We therefore examine again examples of the hidden distributions in Fig. 5.6. They show that the sampler is able to mix very well between the four patterns, for all recovery times. The bad D_{KL} results seem to be indeed due to additional states, which appear similarly often for all recovery times. This confirms the findings from Section 4.2.2 that LIF sampling with alpha-shaped synapses is mixing very well. The results further indicate that the additional reduction of PSP heights with TSO does lead to nearly no improvement of the mixing ability. Examples for the generated samples in Fig. 5.7 show, however, that it has a similar influence on the sample quality as in the case of exponential-shaped synapses. The patterns become weaker with increasing τ_{rec}. We can therefore conclude that using TSO with a high recovery time for LIF sampling with alpha-shaped synapses does not have the advantage of improving mixing, but the disadvantage of reducing the generated sample quality.



(a) Alpha-shaped LIF sampling $\tau_{\text{rec}} = 4.35\,\text{ms}$

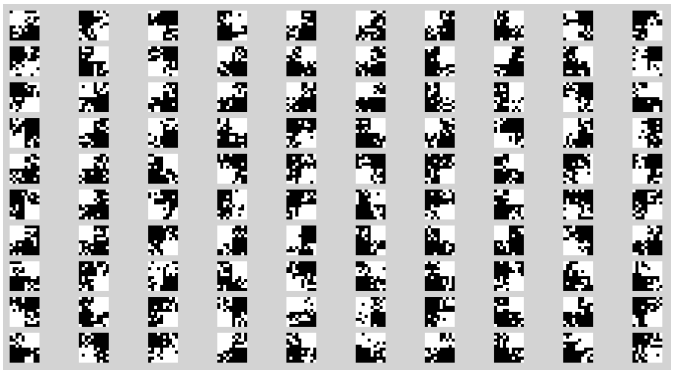


(b) Alpha-shaped LIF sampling $\tau_{\text{rec}} = 6\,\text{ms}$

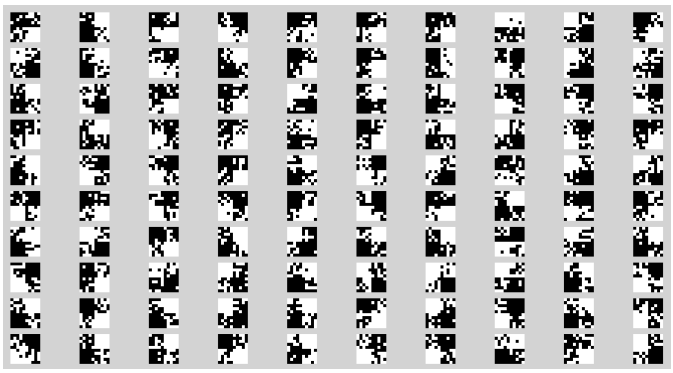


(c) Alpha-shaped LIF sampling $\tau_{\text{rec}} = 8\,\text{ms}$

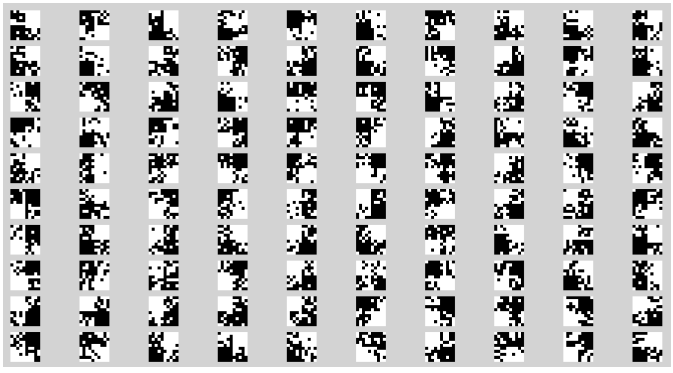
Figure 5.6.: Examples for hidden distributions in the 4 bar example for LIF sampling with CUBA alpha-shaped synapses in analogy to Fig. 5.3.



(a) Alpha-shaped LIF sampling with $\tau_{\text{rec}} = 4.35\,\text{ms}$



(b) Alpha-shaped LIF sampling with $\tau_{\text{rec}} = 6\,\text{ms}$



(c) Alpha-shaped LIF sampling with $\tau_{\text{rec}} = 8\,\text{ms}$

Figure 5.7.: Samples of the visible layer for alpha-shaped LIF sampling with increasing recovery time corresponding to the hidden distributions in Fig. 5.6. The samples should be read from left to right and up to down.

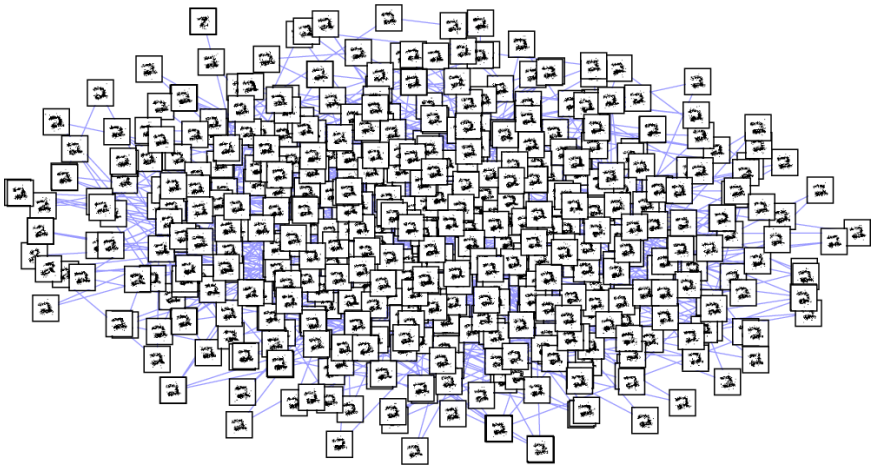
5.3. MNIST 10 Digits

In this section we consider the MNIST example with 10 digits with 28x28 pixels from Section 4.3.2. We use exactly the same trained weights and biases as in Section 4.3.2. We observed, for this example, that just the t-SNE technique (Section 3.10.4) delivers clearly arranged visualizations of the digits. We will therefore, in the following, examine the mixing behavior for increasing values of the recovery time based on the t-SNE method. Figure 5.8 shows the t-SNE results for LIF sampling with exponential-shaped synapses. As we have seen in Section 4.3.2 before, LIF sampling with maintained PSP heights ($\tau_{\text{rec}} = 10\,\text{ms}$) does not mix and generates only samples from one state. The difference to Fig. 4.33a is caused by varying the seed for the random number generator of the Poisson sources. With a higher recovery time of $\tau_{\text{rec}} = 12\,\text{ms}$ mixing is already possible but still difficult, as indicated by the sparse connections between the digit clusters. For $\tau_{\text{rec}} = 14\,\text{ms}$ we can see in Fig. 5.8c already a good mixing ability which is comparable to the LIF sampling result with alpha-shaped synapses in Fig. 4.33b. However, we see a slight decrease in the image quality, similar to Fig. 5.4c for the bar pattern. Increasing the recovery time even further to $\tau_{\text{rec}} = 16\,\text{ms}$ leads to Fig. 5.8d. It shows a good mixing ability as well, but the quality is even worse. This makes it hard to recognize many digits, which leads also to a worse separation in the t-SNE plot. A comparison of a close view of the generated samples between TSO with $\tau_{\text{rec}} = 12\,\text{ms}$ and with $\tau_{\text{rec}} = 16\,\text{ms}$ in Fig. 5.9 demonstrates the decrease of the image quality. It shows that this is an important issue because it makes it difficult to judge which digit the generated samples represent.
From the bar pattern example in the previous section and the fact that LIF sampling with alpha-shaped synapses already mixed well, it seems very likely that using TSO with high τ_{rec} will in this case not improve the mixing behavior as well, but only decrease the quality of the generated samples. Our results indeed confirm this but, as this is not astonishing, we omit them.

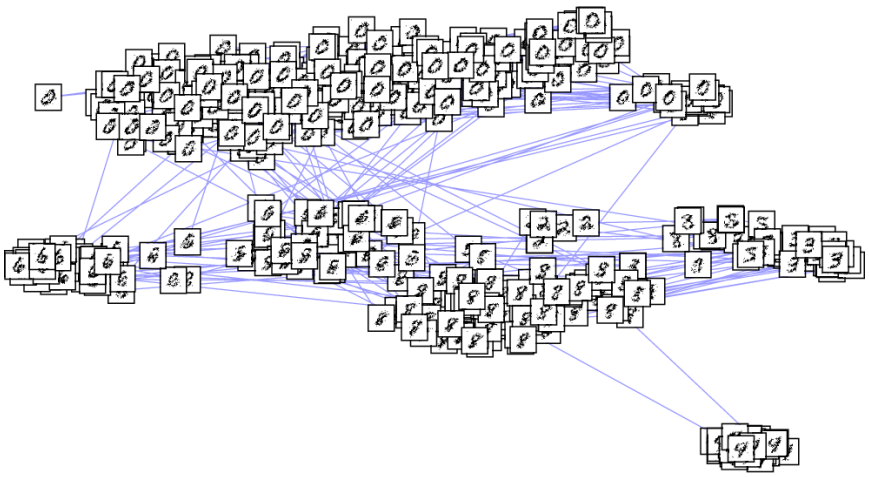
5.4. Balancing Effects

We found by chance an example for a system trained on MNIST where AST and Gibbs sampling generated samples from only one mode, while LIF sampling with exponential-shaped synapses and a high TSO recovery time was still able to mix. These results are astonishing as AST is designed to mix well and in the last chapter we never observed that it fails to do so. One possible explanation is that the underlying distribution in this example was imbalanced, meaning that the probability for one digit to appear was much larger than the probability for all other digits. This would explain why AST didn't mix, because it correctly followed the theoretical distribution and generated samples of the dominant mode. This would, however, mean that LIF sampling with a high recovery time does equalize imbalanced distribution. This observation led to the experiments considered in this section. To reproduce this behavior, we will deliberately train RBMs such that they represent imbalanced distributions. Afterwards, we use LIF sampling with increasing recovery time to generate samples from these distributions. The results

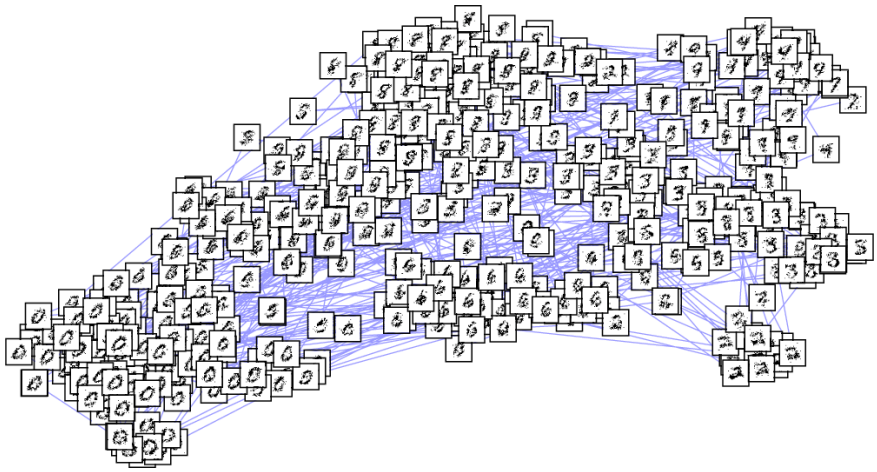
Figure 5.8.: t-SNE visualization for CUBA exponential-shaped LIF sampling



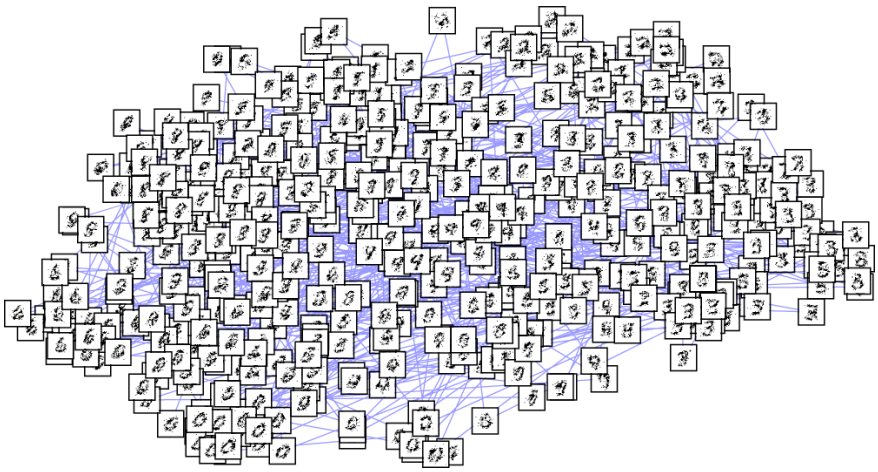
(a) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 10\,\text{ms}$



(b) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 12\,\text{ms}$

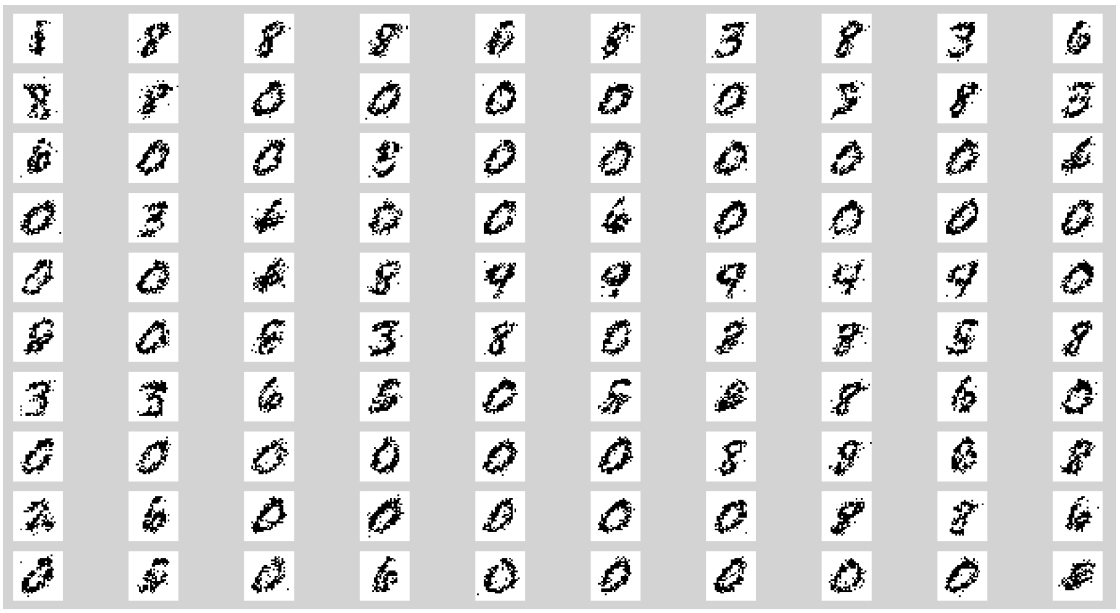


(c) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 14\,\text{ms}$

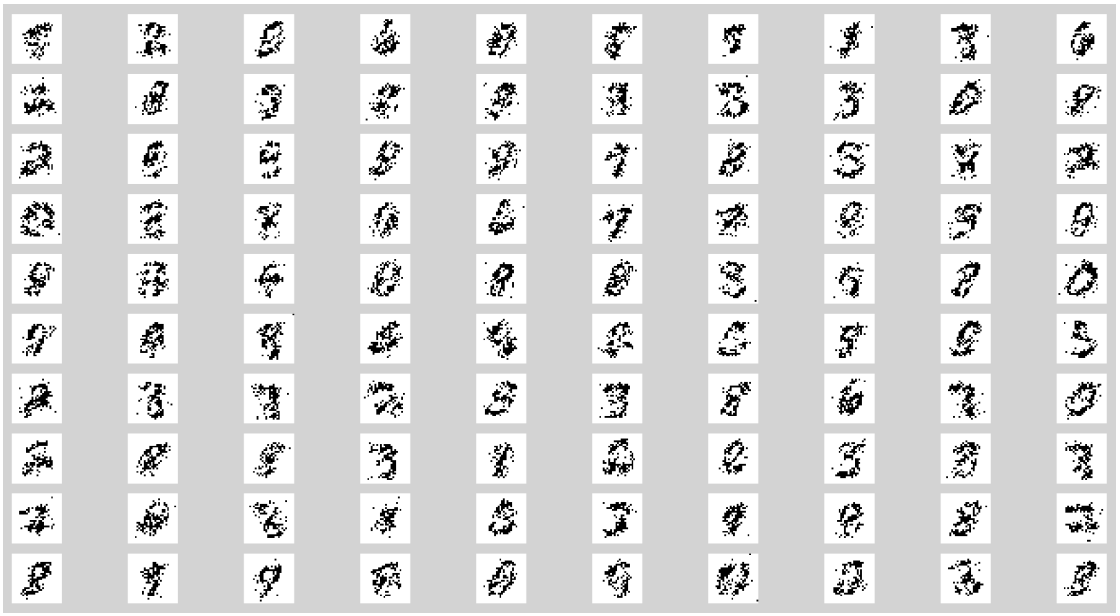


(d) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 16\,\text{ms}$

Figure 5.8.: Result of the t-SNE visualization for CUBA exponential-shaped LIF sampling. We omitted the coordinate axes because they don't add any useful information. We used 28x28 pixel images of the visible layer to mark the mapped samples. The blue lines connect consecutive samples. The used t-SNE parameters are listed in Table A.9. We omitted the COBA results because we didn't observe a significant difference.



(a) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 12\,\text{ms}$



(b) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 16\,\text{ms}$

Figure 5.9.: Samples of the visible layer for CUBA exponential-shaped LIF sampling with recovery times $\tau_{\text{rec}} = 12\,\text{ms}$ and $\tau_{\text{rec}} = 16\,\text{ms}$. The visible units are arranged in 28x28 pixel images as the images in the MNIST data set. The samples should be read from left to right and up to down.

will be compared to the corresponding results for AST to find out whether a balancing of the distributions has happened.

5.4.1. Imbalanced MNIST with 3 Digits

In this section we consider an RBM trained on the same 3 digit data set as in Section 4.3.1. As before we use 144 visible and 10 hidden units. The reason why we are considering this example again is because it is small enough to compute the theoretical hidden distribution. This makes it possible to observe whether a trained RBM represents an imbalanced distribution. Choosing many training steps with a relatively large learning rate with PCD makes it furthermore very likely to obtain imbalanced distributions. The learning parameters for the following example are listed in Table A.7. They yield an imbalanced distribution, as can be seen in Fig. 5.11a for AST. As in Section 4.3.1, we can judge from the receptive fields of the hidden states (Fig. 5.12) the digit class of the most likely states. This shows that "0" is the dominant digit in this distribution. The other two digits have a significantly lower probability to appear. AST seems to represent this distribution very well. A look at the star plot for AST in Fig. 5.10 confirms that also the samples follow this behavior and mainly represent the "0".

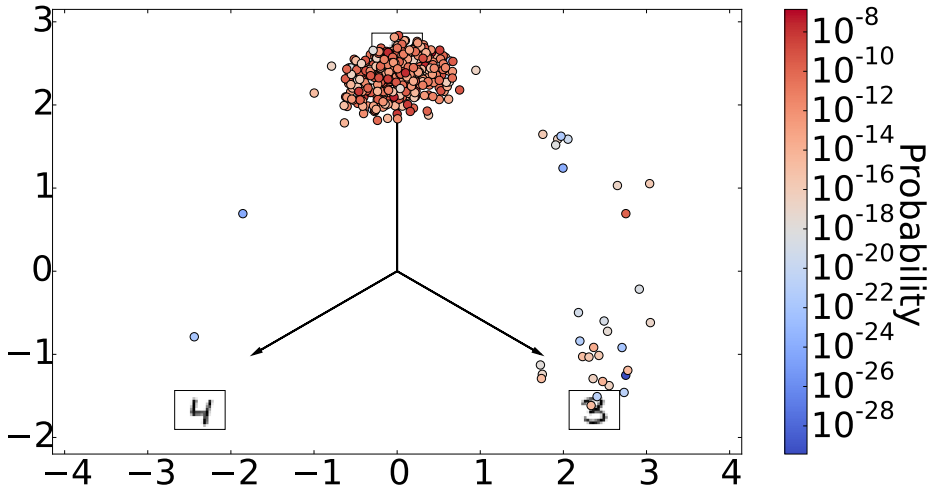
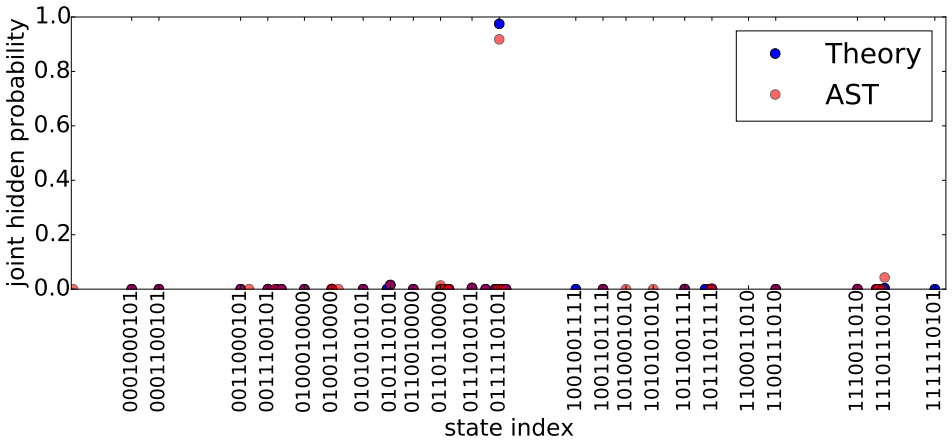
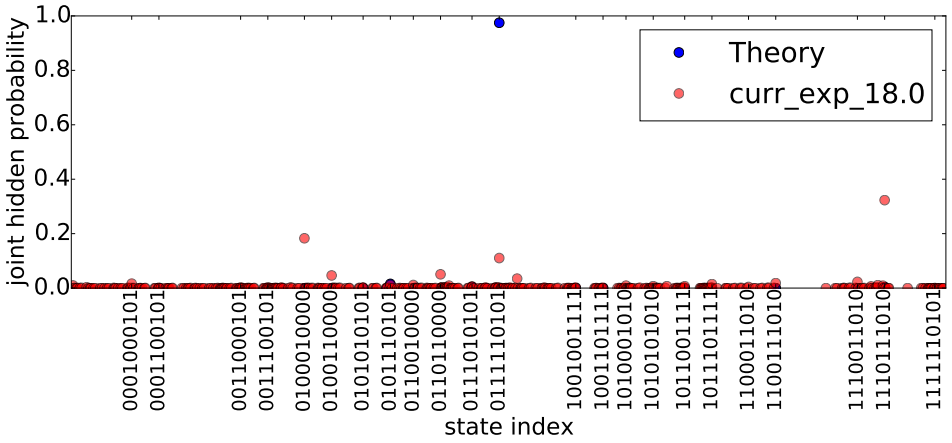


Figure 5.10.: Star plot for AST. The three pictures of a "0", "3" and a "4" denote the axes which represent them. To obtain a clear plot we just showed every 100^{th} sample leading in total to 500 samples. The colors correspond to the probabilities of the samples to occur.



(a) AST



(b) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 18\,\text{ms}$

Figure 5.11.: Hidden distributions for the 3 digit example for AST and CUBA exponential-shaped LIF sampling with $\tau_{\text{rec}} = 18\,\text{ms}$. Displayed is the theoretical and the sample distribution after $5 \cdot 10^5\,\text{ms}$ ($5 \cdot 10^4$ sampling steps). On the x-axis just the non-overlapping hidden states which have one of the 100 largest theoretical probability values are listed. The y-axis represents the probabilities of the hidden states.

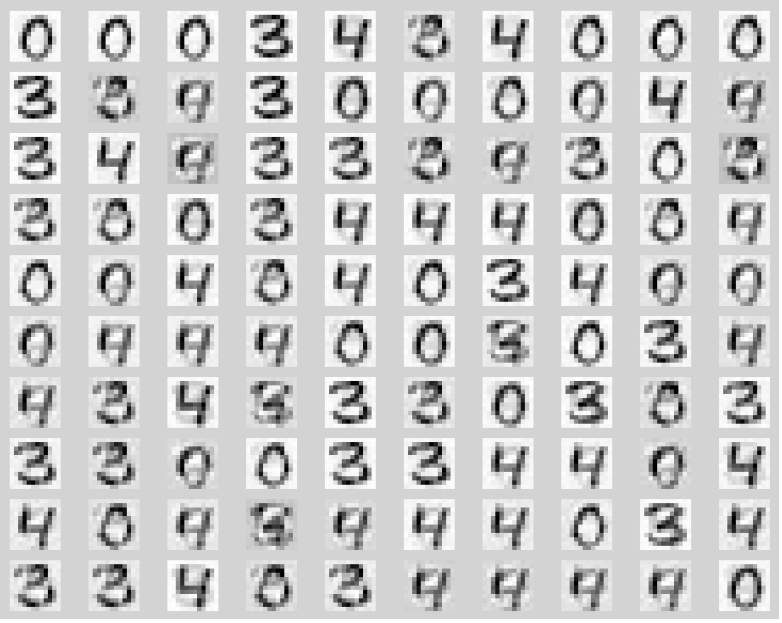


Figure 5.12.: Receptive fields of the 100 hidden states with the largest probabilities in Fig. 5.11. To determine them, we calculated the mean receptive field of all active hidden units.

The corresponding star plot for LIF sampling with exponential-shaped synapses (Fig. 5.13) exhibits indeed a balancing of the distribution for increasing recovery time. However, we can also see that many intermediate states appear, which connect the different clusters. This is due to the mixing behavior of TSO, illustrated in the image series in Fig. A.3, which gradually moves between modes by passing intermediate samples. It seems that especially for these imbalanced distributions LIF sampling needs to take more intermediate samples to mix between the modes. The hidden distribution (Fig. 5.11b) for LIF sampling with $\tau_{\text{rec}} = 18\,\text{ms}$ shows that the hidden layer follows the behavior of the samples leading to a balanced hidden distribution as well. The sampled states (Fig. 5.14) demonstrate, as in Section 5.3, the reduction of the image quality for increased recovery time. One can also see many samples which look like mixtures between two of the digits. They correspond to the intermediate states observed in the star plot.
The balancing effect can be observed similarly for LIF sampling with alpha-shaped synapses, which is demonstrated in the star plots in Fig. 5.15. However, we observe here the same reduction of the image quality as in the exponential-shaped case. Figure 5.15 furthermore shows that the balancing effect does not occur for maintained PSP heights ($\tau_{\text{rec}} = 4.35\,\text{ms}$). This shows that the better mixing property of the alpha-shaped model, which we have observed in the last chapter, is decoupled from the balancing effect. The results rather indicate that the balancing effect is solely due to the TSO mechanism with high recovery times.

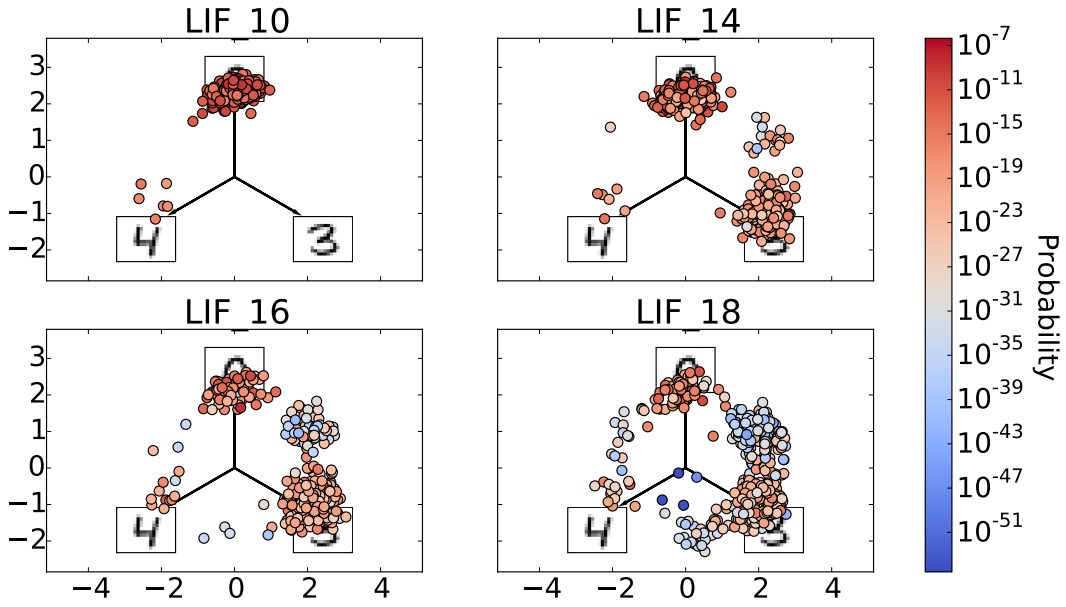
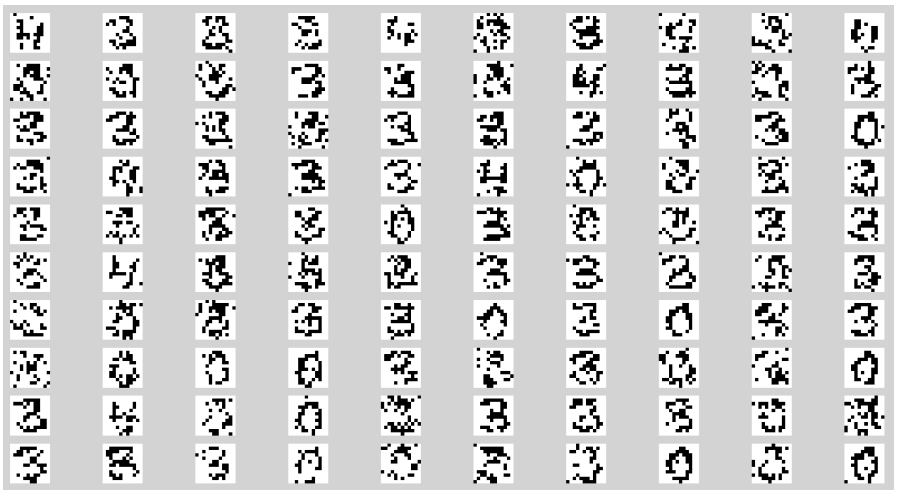


Figure 5.13.: Star plots for CUBA LIF sampling with exponential-shaped synapses and increasing recovery time from $\tau_{\text{rec}} = 10\,\text{ms}$ to $\tau_{\text{rec}} = 18\,\text{ms}$. We omitted the results for COBA LIF sampling because we didn't observe a significant difference.

(a) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 10\,\text{ms}$



(b) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 18\,\text{ms}$

Figure 5.14.: Samples of the visible layer for exponential-shaped LIF sampling for different recovery times. The visible units are arranged in 12x12 pixel images as the learned MNIST digits. For each sampler 100 samples taken equally distributed over the sampling time are shown. The samples should be read from left to right and up to down.

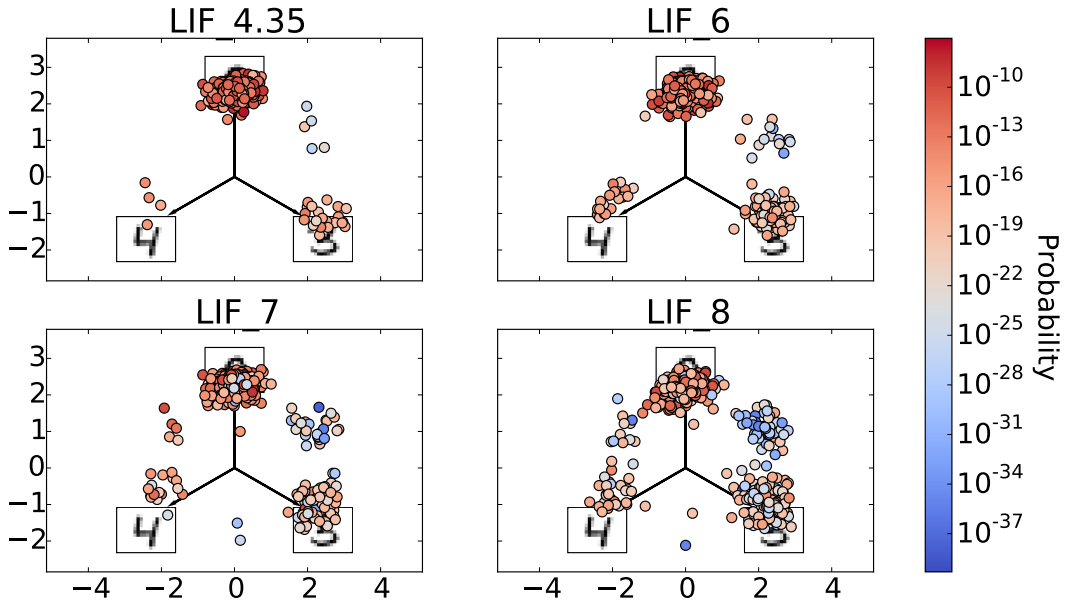


Figure 5.15.: Star plots for CUBA LIF sampling with alpha-shaped synapses and increasing recovery time from $\tau_{\text{rec}} = 4.35\,\text{ms}$ to $\tau_{\text{rec}} = 8\,\text{ms}$. We omitted the results for COBA LIF sampling because we didn't observe a significant difference.

5.4.2. Imbalanced MNIST with 10 Digits

In this section we want to investigate the balancing effect for larger RBMs trained on all 10 MNIST digits. To obtain an imbalanced trained distribution we will already start with an imbalanced training data set (Fig. 5.16). In Section 4.3.2 we have seen that mixing is an important issue in 10 digit MNIST examples with 28x28 pixels but just a small one for 12x12 pixels. As we want to only investigate the balancing effect we use 12x12 pixel images in the training data set to avoid a large influence of the mixing issue. Hence, we use an RBM with 144 visible units. For the hidden layer we choose 100 units to have a large enough model for the training data set. The used learning parameters for the following example are listed in Table A.8. As in Section 5.3 we will use t-SNE to interpret the sampling results. Figure 5.17 shows the t-SNE result for AST. It demonstrates that AST represents the learned distribution very well. It mainly generates samples of the different kinds of "2"s and just rarely of other digit classes. Figure 5.18 exhibits the corresponding results for LIF sampling with exponential-shaped synapses and increasing recovery time. The visualization for $\tau_{\text{rec}} = 10\,\text{ms}$ shows that LIF sampling with maintained PSP heights does nearly solely generate samples from the dominant "2" mode. The sparse connections between the clusters exhibit, as we have seen before (Section 4.3.2), that the sampler has also mixing issues. This is probably the reason why the samples from the other modes are missing.

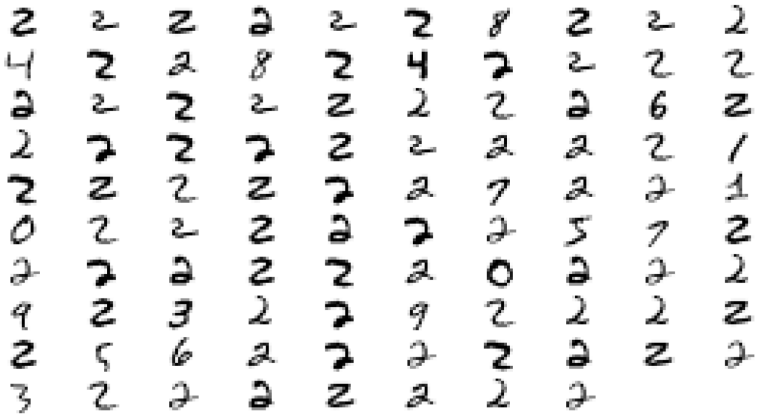


Figure 5.16.: Imbalanced training data set. It contains 80 examples of a "2" and 2 examples for all other digits respectively. The images are reduced two 12x12 pixels.

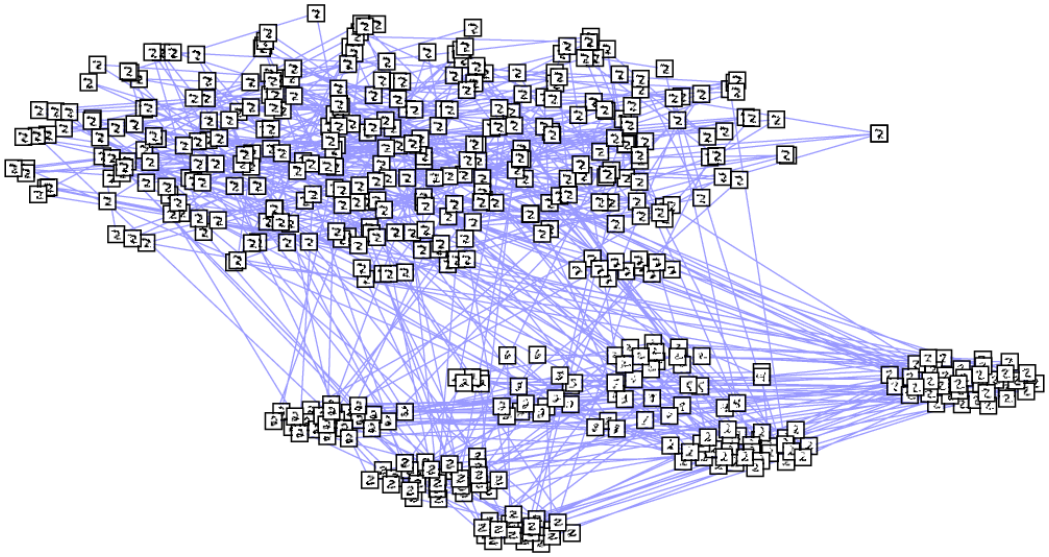
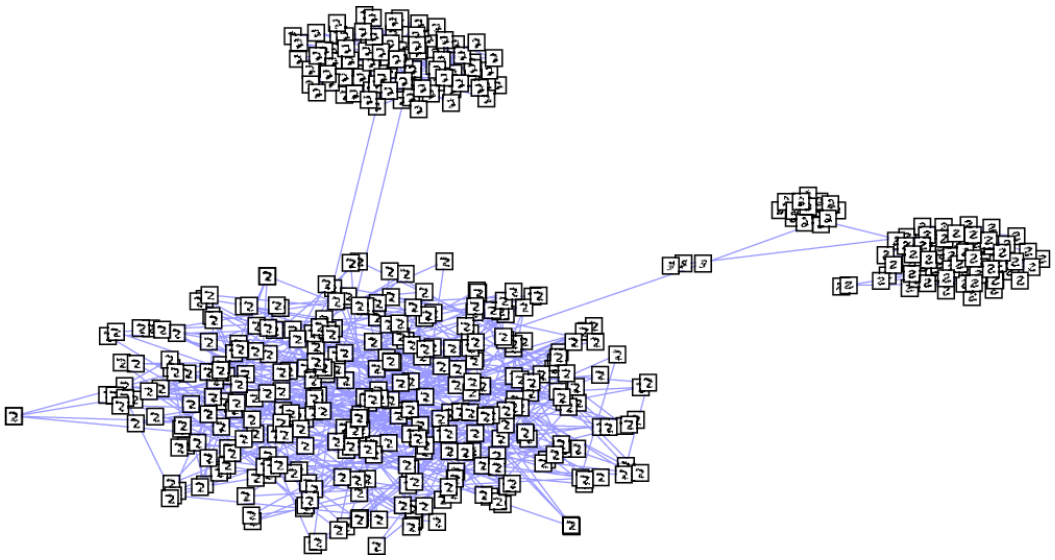


Figure 5.17.: Result of the t-SNE visualization for AST. We used images of the visible layer to mark the mapped samples. The blue lines connect consecutive samples. The used t-SNE parameters are listed in Table A.9.

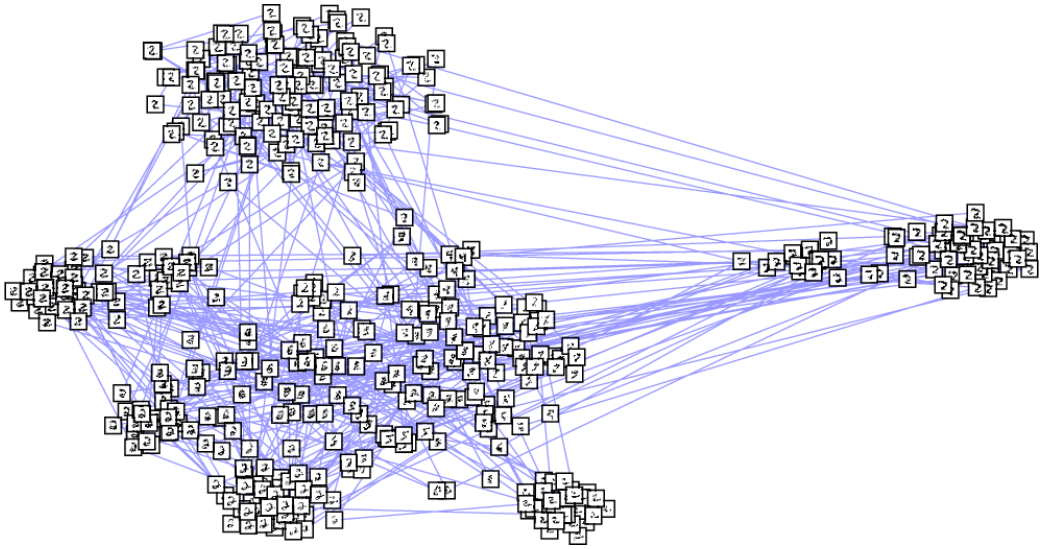
An increased recovery time of $\tau_{\text{rec}} = 13\,\text{ms}$ already exhibits a balancing effect (Fig. 5.18b). This can be seen on the much larger fraction of samples from the minor modes compared to the AST result in Fig. 5.17. However, the "2" mode is still dominant. For $\tau_{\text{rec}} = 15\,\text{ms}$

we see a stronger balancing (Fig. 5.18c), where approximately only a half of the samples still represent the dominant mode. In between there are many samples from minor modes but also many unclear samples. We observed this as well for the 3 digit example in the previous section. These unclear samples are probably intermediate samples along the paths the sampler has to take to mix between the modes. A comparison of the generated samples between LIF sampling with $\tau_{\text{rec}} = 10\,\text{ms}$ and LIF sampling with $\tau_{\text{rec}} = 15\,\text{ms}$ (Fig. 5.19) demonstrates the reduction of the sample quality. A comparison to the 3 digit example in the previous section (Fig. 5.14) shows that the image quality is much worse, although we used a smaller recovery time of $\tau_{\text{rec}} = 15\,\text{ms}$.
The t-SNE results for alpha-shaped LIF sampling are shown in Fig. 5.20. For maintained PSP shapes ($\tau_{\text{rec}} = 4.35\,\text{ms}$) we see, as in Section 4.3.2, that the alpha-shaped model has no mixing issues. It therefore correctly reproduces the training data set with many dominant "2" samples and only few samples of other modes. For increasing recovery time it behaves similar to the exponential-shaped model. It increasingly balances the distribution which can be seen on the larger fraction of samples of minor modes. However, it exhibits the same quality reduction of the generated samples. This is in accordance with our previous results for the 3 digit example (Fig. 5.15).

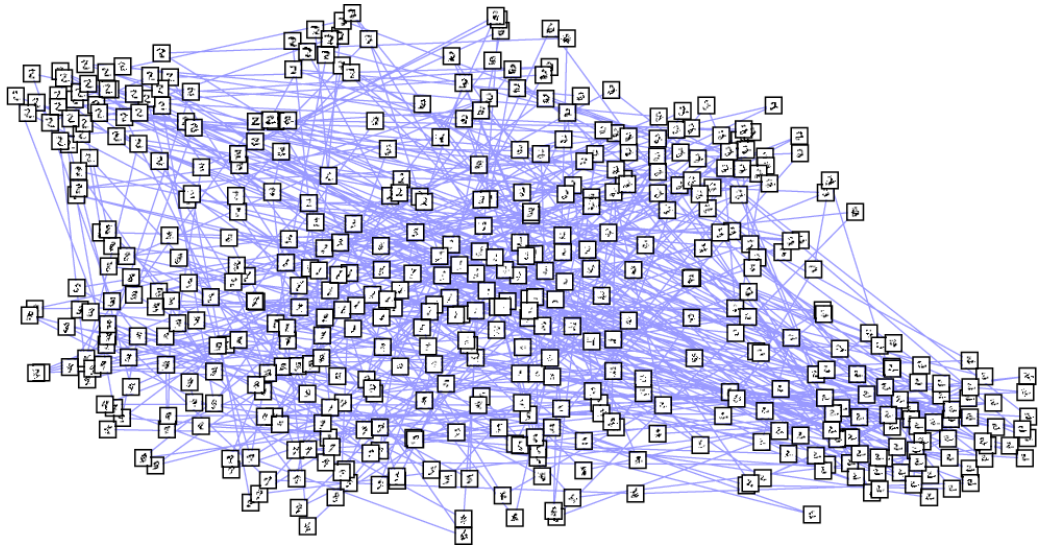
Figure 5.18.: t-SNE visualization for exponential-shaped LIF sampling



(a) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 10\,\text{ms}$

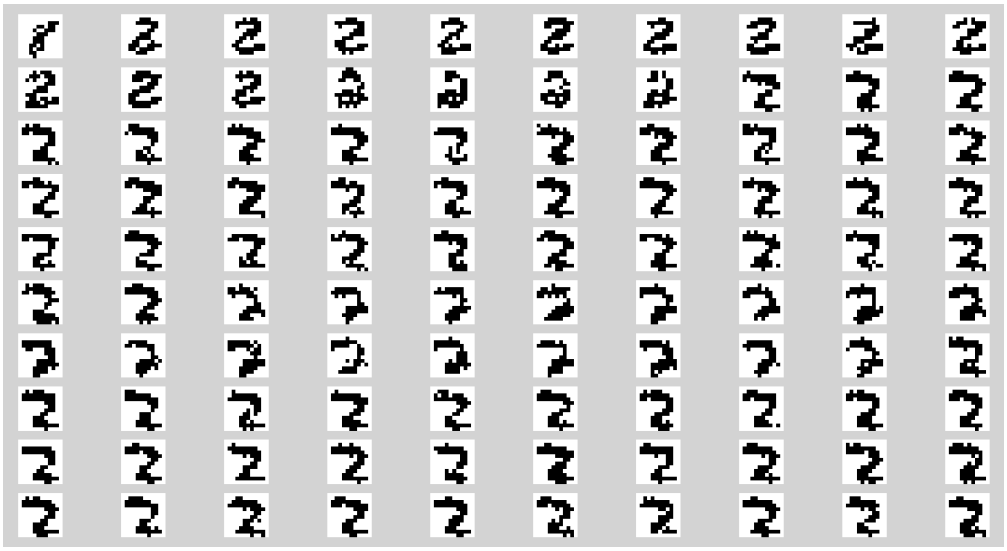


(b) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 13\,\text{ms}$

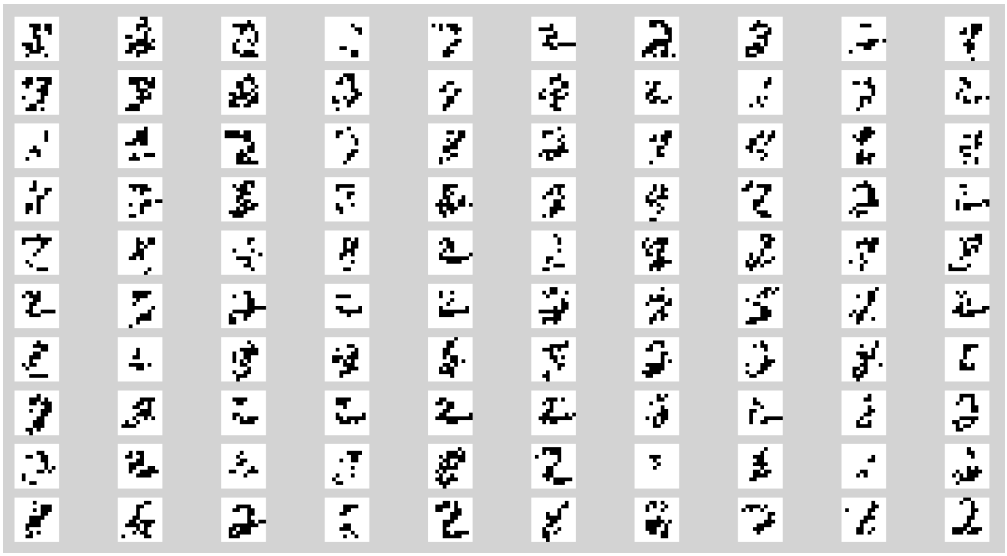


(c) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 15\,\text{ms}$

Figure 5.18.: Result of the t-SNE visualization for CUBA exponential-shaped LIF sampling. The used t-SNE parameters are listed in Table A.9. The corresponding COBA results are similar and therefore omitted.



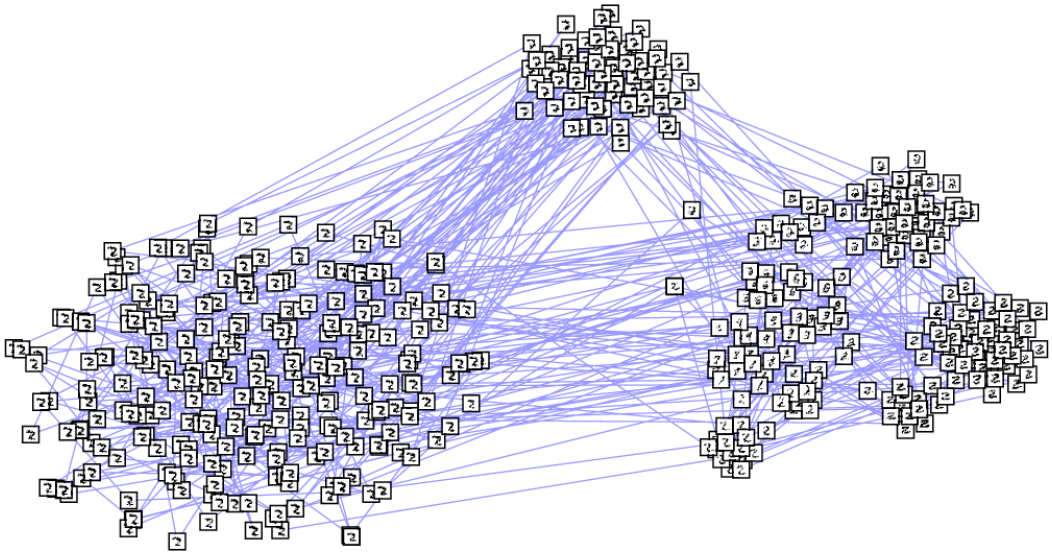
(a) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 10\,\text{ms}$



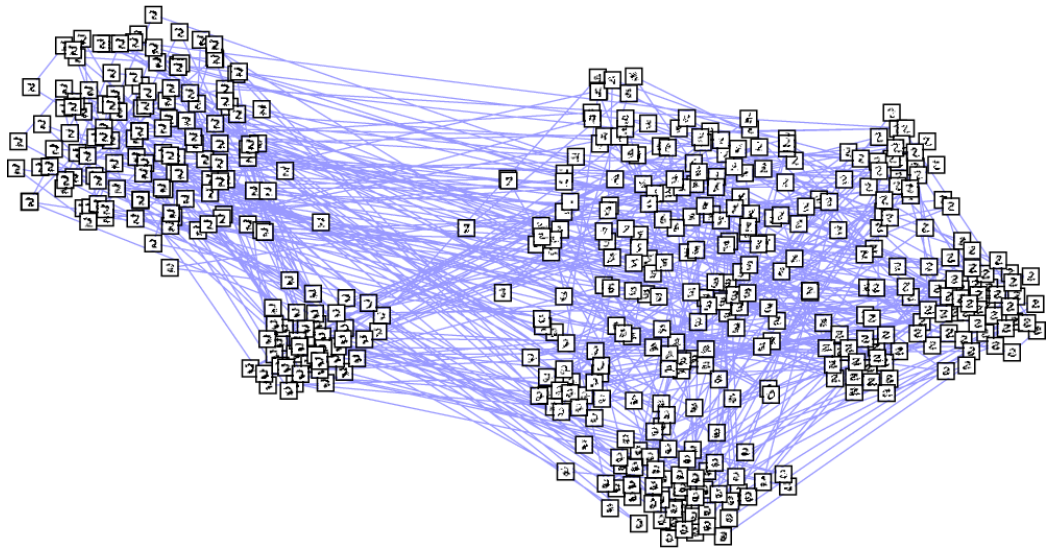
(b) Exponential-shaped LIF sampling with $\tau_{\text{rec}} = 15\,\text{ms}$

Figure 5.19.: Samples of the visible layer for exponential-shaped LIF sampling with different recovery times. The visible units are arranged in 12x12 pixel images as the learned MNIST digits. For each sampler 100 samples taken equally distributed over the sampling time are shown. The samples should be read from left to right and up to down.

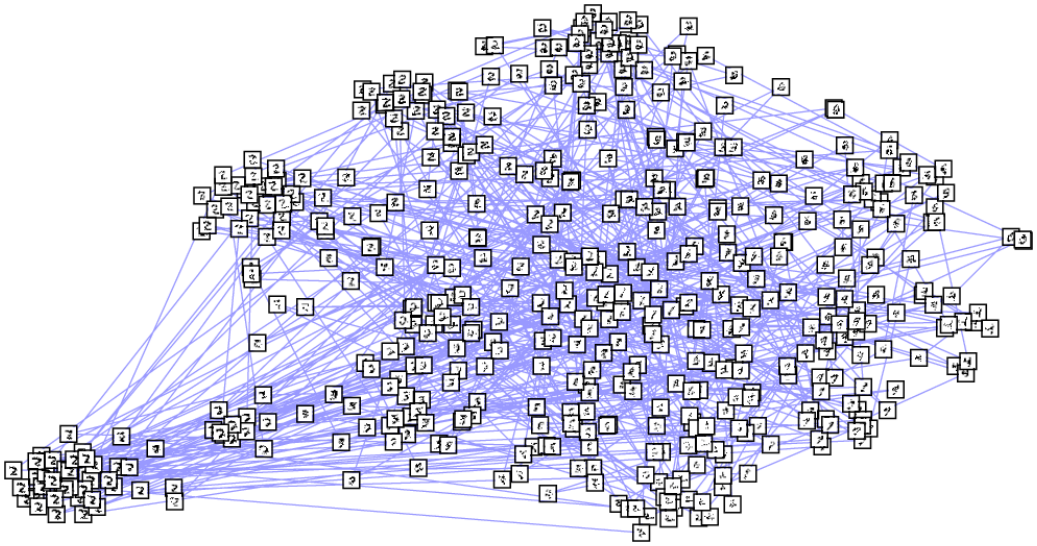
Figure 5.20.: t-SNE visualization for CUBA alpha-shaped LIF sampling



(a) Alpha-shaped LIF sampling with $\tau_{\text{rec}} = 4.35\,\text{ms}$



(b) Alpha-shaped LIF sampling with $\tau_{\text{rec}} = 5.0\,\text{ms}$



(c) Alpha-shaped LIF sampling with $\tau_{\text{rec}} = 6.0\,\text{ms}$

Figure 5.20.: Result of the t-SNE visualization for CUBA alpha-shaped LIF sampling. The used t-SNE parameters are listed in Table A.9. The corresponding COBA results are similar and therefore omitted.

5.5. Discussion

We observed in this section that increasing the recovery time of the TSO mechanism beyond the value necessary to maintain the PSP heights leads to two different effects in LIF sampling: an improved mixing and a balancing effect.

The mixing effect was examined for the bar pattern (Section 5.2) and the balanced 10 digit MNIST example (Section 5.3). In both examples we saw that LIF sampling with exponential-shaped synapses exhibits an improved mixing when increasing the recovery time. On the other hand, LIF sampling with alpha-shaped synapses showed no improvement beyond its already good mixing performance for maintained PSP heights. However, for both sampling procedures we observed a reduction of the quality of generated samples. This reduction is more severe for the MNIST example, than for the bars, as for larger recovery times it became hard to recognize many of the generated digits. The reason of the improved mixing was already explained in Section 5.1. It is due to the reduction of the synaptic weights of neurons, which are constantly firing, for higher recovery times. This leads to a reduction of the probability in regions of the state space where the sampler gets stuck, which is illustrated in the image sequence in Fig. A.3. This mixing mechanism explains also the observed reduced image quality.

When the weight connections between the hidden and visible neurons, representing a pattern, become slowly reduced, it becomes more likely that certain neurons, belonging to that pattern, turn off. This starts a chain reaction, because in the next sampling step these deactivated neurons will stop activate other neurons of that pattern and also stop deactivating neurons which do not belong to the pattern. As a consequence, more and more neurons belonging to the pattern will be deactivated and other neurons activated. This leads to the noisy and weak patterns for high recovery times. As the patterns in the MNIST data set are weaker than the bar patterns, this explains the more severe consequences for MNIST.
Nevertheless, the examples showed that, when choosing the recovery time carefully, the TSO mechanism can help to overcome the missing mixing ability for LIF sampling with exponential-shaped synapses. It could therefore be used to improve mixing in large systems on future implementations of LIF-based BMs on HICANN, which does currently only realize exponential-shaped synapses. For LIF sampling with alpha-shaped synapses, however, the TSO mechanism should only be used to maintain the PSP heights. Higher recovery times lead in this model to a worse quality of generated samples and do not significantly improve mixing.

The balancing effect was demonstrated for two examples of imbalanced distributions trained on the MNIST data set. One small system with 3 MNIST digits (Section 5.4.1) and a larger system with 10 MNIST digits (Section 5.4.2). LIF sampling with both alpha and exponential-shaped synapses showed a balancing of the distributions for both examples. However, as for mixing, this comes together with a reduction of the image quality, which was worse in the 10 digit example than in the 3 digit one.
The mechanism for the balancing effect is the same as for mixing. The difference is that in the balancing examples we started with strongly imbalanced distributions. For these distributions LIF sampling will spend most of the time in the dominant mode. As a consequence, due to the TSO mechanism, this mode will be mainly affected by the reduction. However, as the other modes are rather weak, it will take a longer time for the sampler to find them, which leads to the noisy and unclear intermediate samples, observed in both examples. This effect should become worse for larger state spaces, which explains the worse image quality for the 10 digit example.
The balancing effect could be useful for handling imbalanced data sets, which is an important issue in machine learning and data mining (*He et al.*, 2009; *Chawla*, 2010; *Kubat et al.*, 1997). It could for example be used to reproduce data, which has been learned on only few examples in large imbalanced data sets. This could lead to a better performance in pattern completion tasks for exotic examples. Furthermore, as there is no conceptual difference between data and labels (Section 3.2), the improved pattern completion would also lead to a better classification. However, these conclusions still need to be investigated.

6. Discussion

The aim of this thesis is to investigate the properties of sampling in Boltzmann machines (BMs) based on the leaky integrate-and-fire (LIF) neuron model with a focus on mixing issues.
In Chapter 4 we therefore considered current-based (CUBA) and conductance-based (COBA) LIF sampling with exponential and alpha-shaped synapses. We applied the Tsodyks-Markram (TSO) short-term synaptic plasticity model only to maintain the height of post-synaptic potentials (PSPs) as demanded by the theory. We compared LIF sampling to Gibbs sampling, which is the standard approach in classical BMs. Furthermore, we considered adaptive simulated tempering (AST), which is especially designed to mitigate the mixing issue.
For examples where mixing is not important, like the homogeneous distributions in Section 4.1.1, we observed a close approximation of the theoretical distribution for all samplers. However, LIF sampling showed a saturation where it ceases to approximate the theoretical distribution any further. For the remaining examples, we observed that LIF sampling with exponential-shaped synapses shows the worst ability to mix. The Gibbs sampler performs slightly better but for critical examples, like the bar patterns (Section 4.2.2) or the MNIST example with large image size (Section 4.3.2), it lost its ability to mix as well. Only LIF sampling with alpha-shaped synapses and AST were able to mix in all investigated examples. The image quality for both was similar in most cases, however, for alpha-shaped synapses we observed more intermediate samples which look like mixtures between two digits.
Our explanation for both effects, saturation and different mixing behavior, was the different PSP shape in LIF sampling. Figure 3.7 shows that it exhibits a large variation of potential values during a refractory period compared to the rectangular shape of the abstract sampler. This could facilitate mixing, as it activates neurons in situations where they should, in theory, not be activated or prevents them from being activated. The PSP shape furthermore shows a remaining potential level (tail) after the refractory period. This effect should, in principle, reduce the ability to mix, because the tail acts like a memory of the previous state, biasing the neuron to not change its behavior after the refractory period of the pre-synaptic neuron ends. While for the exponential-shaped model the tail seems to outweigh the variation such that it mixes worse than the abstract rectangular shape, it is the opposite for the alpha-shaped model. Both effects, however, also lead to the saturation in the homogeneous case, because the LIF sampler does not exactly follow the theoretically ideal behavior.
Concerning a possible hardware implementation of LIF sampling on HICANN this result is unfortunate because only exponential-shaped synapses are realized. To improve the mixing behavior also on the currently available hardware, we considered in Chapter 5

a possible approach based on the same TSO model as before. This time, however, we increased the recovery time beyond the value necessary to maintain the PSP heights. The idea behind this approach is to reduce the synaptic influence of neurons which are constantly firing. This leads to a reduction of probability in regions of the state space where the sampler gets stuck, which we have illustrated in the image sequence in Fig. A.3.
We tested this approach on the two examples of Chapter 4 where mixing is critical, namely the bar pattern and the MNIST example with large images size. We saw that this approach works well for LIF sampling with exponential-shaped synapses. It shows an improved mixing when increasing the recovery time of the TSO model. For LIF sampling with alpha-shaped synapses, however, there was no improvement over its already good mixing performance. However, for both synapse models we observed a reduction of the quality of generated samples. We can explain this by the effectively reduced synaptic weights as well, which leads to a weakening of the imprinted patterns. Yet, with a careful choice for the recovery time this approach can be useful to improve mixing for exponential-shaped synapses. As the TSO model is already implemented on HICANN, this is an appealing approach for possible hardware implementations of LIF-based BMs.
While using the TSO model with higher recovery times, we furthermore observed a balancing effect in RBMs representing imbalanced distributions. This effect appears because in these cases LIF sampling spends most of the time in dominant modes. As a consequence, these modes will be mainly affected by the reduction of synaptic weights. Using this effect, it was possible with LIF sampling to reproduce a significant amount of samples from digit classes which were clearly underrepresented in the training data set. This balancing effect could therefore be useful for handling imbalanced data sets, which is an important issue in machine learning (*He et al.*, 2009; *Chawla*, 2010; *Kubat et al.*, 1997). However, for this effect we observed the same reduction of the generated sample quality, limiting its application to moderate choices of recovery times.
For all simulations conducted in this thesis, we didn't observe a significant difference between CUBA and COBA LIF sampling. This is due to our choice of the LIF sampling parameters (Appendix A.2.2). To reduce the nonlinearity in the summation of PSPs for the COBA model (Section 3.6.1), we have chosen a large distance between reversal and resting potentials of the membrane. These values are motivated by *Naud et al.* (2008) and in the range of achievable values of the HICANN system (*Petrovici*, 2015). This demonstrates that the deviations of the COBA model to the theoretically ideal behavior is negligible in the considered examples for a reasonable choice of parameters. This is especially important since the implemented synapse model on HICANN is COBA.

Altogether, we conclude that with LIF sampling, either with alpha-shaped synapses or with exponential-shaped synapses and a high recovery time for TSO, a better mixing behavior than with classical Gibbs sampling is achievable. However, using alpha-shaped synapses is advised in terms of neural sampling performance, as they show improved mixing without extra effort and no disadvantages as, for example, a reduced image quality. These results therefore indicate that an implementation of alpha-shaped synapses on a future version of HICANN should be considered.

7. Outlook

Using TSO with high recovery times, as proposed in this thesis, is a promising approach to improve mixing in LIF sampling on the next version of the HICANN chip, which implements exponential-shaped synapses. Beside that, future implementations of HICANN which may include the alpha-shaped synapse model as well would directly benefit from the improved mixing demonstrated in this thesis for this synapse model.

One approach to mitigate the quality reduction for generated samples, which accompanies the TSO approach, could be using a combination of facilitation and depression for the TSO model. This leads, like in Fig. 7.1, to a short buildup of the PSPs and afterwards a faster decay. The buildup could strengthen patterns for a short time and the decay induce faster transitions to other modes, such that less intermediate samples are generated. Testing this mechanism is currently in progress. However, as on the next version of HICANN no combination of facilitation and depression is possible, this would be applicable on future versions of the chip only, which may include this possibility.

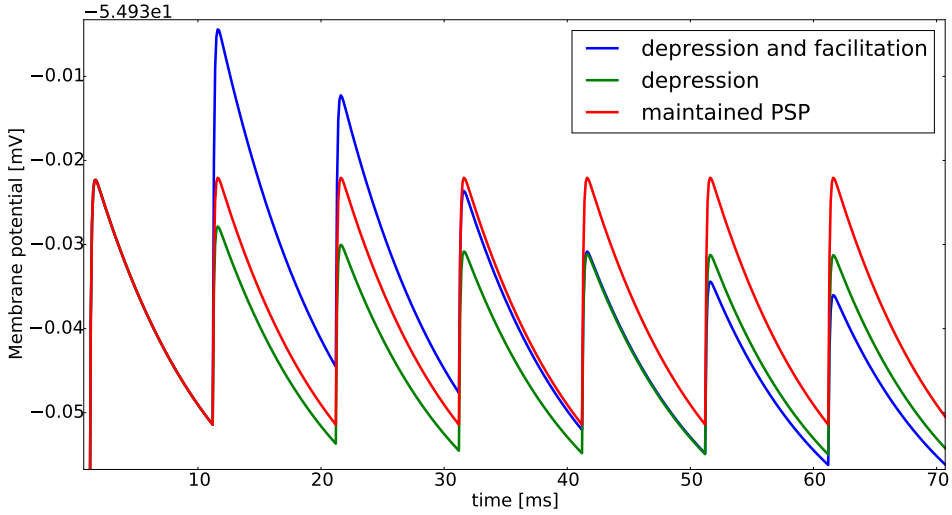


Figure 7.1.: Simulation of the membrane potential of a neuron which receives a spike every refractory period with refractory time $\tau_{\text{ref}} = 10\,\text{ms}$ for exponential-shaped synapses. For the red curve we used TSO only to maintain the PSP height ($\tau_{\text{rec}} = 10\,\text{ms}$). For the green curve we applied $\tau_{\text{rec}} = 14\,\text{ms}$ to reduce the PSPs. For the blue curve we also used the facilitation mechanism of TSO (Section 3.7) with $U_{SE} = 0.4$, $\tau_{\text{facil}} = 50.0\,\text{ms}$ and $\tau_{\text{rec}} = 50.0\,\text{ms}$.

The large difference of the mixing behavior between alpha and exponential-shaped synapses demonstrates that the PSP shape has a huge influence on mixing. It could therefore be of interest to test the mixing ability for other PSP shapes by varying the synaptic time constants or by applying, for example, a difference of exponentials as synapse model. With these experiments we could also test our proposed explanation for the different mixing behavior of alpha and exponential-shaped synapses. According to our explanation, PSP shapes with a large variation of potential values should exhibit a good mixing, while PSP shapes with larger potential tales after the refractory period should perform worse.

In this thesis, only the influence of mixing on sampling was considered. It has already been shown that better mixing is important for training classical RBMs (*Welling et al.*, 2003; *Tieleman and Hinton*, 2009; *Salakhutdinov*, 2009). With the STDP-based contrastive divergence approach, demonstrated in *Weilbach* (2015), the improved mixing with TSO could therefore be used to improve training in LIF-based RBMs.

Another possible approach to improve mixing could be to mimic the behavior of AST in LIF sampling. Equivalent to the temperature increase in classical sampling, one would be increasing the rates of the background Poisson sources in LIF sampling. A crude approximation of AST, which might already improve mixing, could be to periodically modulate the background Poisson rates with a fixed pattern (e.g. linear rise, short halt at the highest rate and linear decay). Doing this periodically has the advantage of knowing the time ranges with higher rates in advance. These should then be excluded from the evaluation afterwards. With this, only samples from the normal temperature (background rate) can be counted, avoiding intermediate samples.
Furthermore, for classical BMs there is also an approach to use AST for learning called *coupled adaptive simulated tempering* (CAST) (*Salakhutdinov*, 2010). Similarly to the TSO mechanism, we can combine the LIF approximation of AST with STDP-based contrastive divergence to obtain an LIF-based version of the CAST algorithm.

For the balancing effect accompanying TSO with high recovery times we concluded that it could be used to reproduce sparsely represented data in LIF-based RBMs trained on imbalanced data sets. The ability to complete patterns and classify input based on this data could benefit from that. However, these effects still need to be investigated. A possible experiment could be, similar to Section 5.4.2, an RBM trained on ten MNIST digits, with one digit, for example “1”, being dominant in the training data. Now we can clamp the neurons of the visible layer such that generated samples will always show a circle in the lower part of the image. Performing pattern completion could lead to samples of a mixture between the circle and the upper part of the “1” in an RBM without balancing. This is due to the probabilities of the other modes still being too low for the sampler to reach them. In an RBM with balancing, however, sampling would more often lead to the expected samples of a “6” or an “8”.

Our final goal is to implement LIF-based BMs on neuromorphic hardware, starting with

the HICANN chip. Until then there are, however, still several issues to be addressed. For LIF sampling we assumed that every neuron receives input from at least two Poisson sources. Networks which are capable of, for example, learning MNIST require therefore a large number of uncorrelated noise sources. Such a large amount of noise sources is currently not implemented on HICANN. One approach to solve this could be the use of other neural networks running on the same device as noise sources for the sampling network. These so-called *sea-of-noise* networks are a topic of active investigation (*Jordan et al.*, 2014).
Another issue is the so-called *fixed-pattern noise*, which is a parameter variability from neuron to neuron as well as from synapse to synapse on the hardware. It is caused by the manufacturing process of the neuromorphic hardware device and cannot be avoided. To account for it in our current software simulations, one can add, as in *Probst et al.* (2015), Gaussian noise to the neuron and synapse parameter during the initialization of the network. However, the noise affecting the neuron parameters that are not changed when setting weights and biases can be completely absorbed into the translation rules from Section 3.8.
Furthermore, the influence of the 4-bit weight resolution of the HICANN chip on LIF-based BMs still has to be investigated.
In return, the advantages of a well working implementation of algorithms like LIF-based BMs on neuromorphic hardware are tremendous. The inherent parallelism would allow scaling the network size to thousands of neurons without increasing the emulation time. This allows efficient training and sampling for the full MNIST or even larger real world data sets and finally also for real time data, obtained, for example, with silicon retinas (*Lichtsteiner et al.*, 2008) or cochleas (*Liu and Delbruck*, 2010). Combined with a low power consumption per synaptic interaction these algorithms could operate on mobile devices in areas where analysis and fast predictions based on real time data is essential. Examples would be self-driving cars, speech recognition, quality assurance in industrial production facilities or sensors in data-intensive scientific research or for the analysis of clinical data to predict diseases.

A. Appendix

A.1. Acronyms

API Application Programming Interface

ASIC Application-Specific Integrated Circuit

AST Adaptive Simulated Tempering

AdEx Adaptive-Exponential Integrate-and-Fire

BM Boltzmann Machine

CD Contrastive Divergence

COBA COnductance-BAsed

CUBA CUrrent-BAsed

DNC Digital Network Chip

FACETS Fast Analog Computing with Emergent Transient States

FPGA Field Programmable Gate Array

HCS High Conductance State

HICANN High Input Count Analog Neural Network

KL Kullback-Leibler

LIF Leaky Integrate-and-Fire

MCMC Markov Chain Monte Carlo

ODE Ordinary Differential Equation

OU Ornstein-Uhlenbeck

PCA Principal Component Analysis

PCD Persistent Contrastive Divergence

PSP Post-Synaptic Potential

PSR	Post-Synaptic Response
RBM	Restricted Boltzmann Machine
SNE	Stochastic Neighbor Embedding
STP	Short-Term Plasticity
t-SNE	t-Distributed Stochastic Neighbor Embedding
TSO	TSOdyks-Markram (synaptic plasticity model)
VLSI	Very-Large-Scale Integration

A.2. Parameter

A.2.1. Adaptive Simulated Tempering

For AST we used throughout this thesis:

K	10	number of temperature levels
β_K	0.1	minimal inverse temperature
$\gamma^{(0)}$	90.0	initial weight adaption factor

Table A.1.: Adaptive Simulated Tempering Parameter.

The K inverse temperature levels are linearly distributed within the range [1.0, β_K]. The weight adaption factors decay during sampling according to

$$\gamma^{(\tau)} = \frac{\gamma^{(0)}}{100 + \tau} \quad , \tag{A.1}$$

where τ denotes the current sampling step.

A.2.2. LIF Sampling

We used sbs 1.3.0 (Section 2.2.2), together with Pynn 0.8 (Section 2.1) and Nest 2.4.2 (Section 2.2.1) for LIF sampling throughout this thesis. We have chosen the same parameter for COBA and CUBA LIF sampling. The only difference is that the excitatory and inhibitory reversal potentials can be neglected for CUBA LIF sampling. For the simulations we use a time step of $\mathrm{d}t = 0.1\,\mathrm{ms}$ and a synaptic delay of $\tau_{\mathrm{delay}} = 0.1\,\mathrm{ms}$. For the exponential-shaped LIF neuron and synapse model, we have chosen the following parameters:

C_m	0.2 nF	membrane capacitance
τ_m	0.1 ms	membrane time constant
$E_{\text{exc}}^{\text{rev}}$	0 mV	excitatory reversal potential
$E_{\text{inh}}^{\text{rev}}$	-100 mV	inhibitory reversal potential
ϑ	-50 mV	threshold voltage
u_{rest}	-50 mV	resting potential
ρ	-50.01 mV	reset potential
$\tau_{\text{exc}}^{\text{syn}}$	10 ms	excitatory synaptic time constant
$\tau_{\text{inh}}^{\text{syn}}$	10 ms	inhibitory synaptic time constant
τ_{ref}	10 ms	refractory time constant
I_{offset}	0 nA	offset current

Table A.2.: Neuron and synapse parameters used for exponential-shaped LIF sampling.

For the alpha-shaped LIF sampling we use the same parameters except that we have chosen for the synaptic time constants $\tau_{\text{exc}}^{\text{syn}} = \tau_{\text{inh}}^{\text{syn}} = 2.6\,\text{ms}$. A sweep over the synaptic time constant for which we have evaluated the performance of alpha-shaped LIF sampling to reproduce a homogeneous random distribution led to this value (see Fig. A.1). For the parameter translation between abstract and LIF regime (calibration) we use one excitatory and one inhibitory Poisson source for each neuron with an input rate of $\nu = 400\,\text{Hz}$ and a weight connection of $w_{\text{poisson}} = 0.002\,\mu\text{S}$.

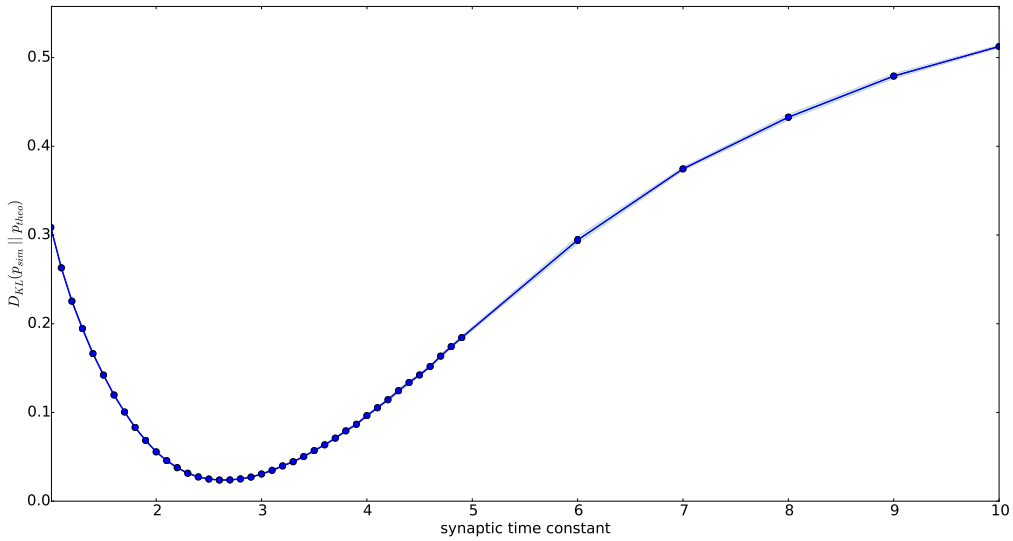


Figure A.1.: Sampling performance measured with the Kullback-Leibler divergence over the synaptic time constant. We considered the theoretical distribution of a fully-connected Boltzmann machine with 5 units. The weights and biases have been drawn from a Gaussian distribution with mean $\mu = 0$ and standard deviation $\sigma = 2.0$. The sample distribution is obtained through LIF sampling with CUBA alpha-shaped synapses for 10^6 ms for each data point. Except of the varied synaptic time constant the parameters were identical to Table A.2. Displayed is the mean of the Kullback-Leibler divergence between the theoretical and sample distribution over 5 runs where the sampling seed has been varied. The colored area shows the standard deviation. The minimum is located at $\tau_{\text{syn}} = 2.6\,\text{ms}$.

A.2.3. Tsodyks-Markram Model

For LIF sampling in Chapter 4 we chose for the Tsodyks-Markram model the parameters listed in Table A.3. These parameters were chosen to maintain the PSP heights for the corresponding synapse models as demonstrated in Fig. 3.5. For LIF sampling in Chapter 5 we varied the recovery time constant but kept the other parameters fixed.

U_0	1.0	utilization of synaptic efficacy
$\tau_{\text{rec}}^{\text{exp}}$	10 ms	recovery time constant for exponential-shaped synapses
$\tau_{\text{rec}}^{\text{alpha}}$	4.35 ms	recovery time constant for alpha-shaped synapses
τ_{facil}	0 ms	relaxation time constant of the facilitation

Table A.3.: Tsodyks-Markram model parameter.

A.2.4. Learning

To train the 3 digit example in Section 4.3.1 we used:

learning algorithm	PCD
number of visible units	144
number of hidden units	10
batch size	3
training steps	99830
learning rate $\eta(t)$	$\frac{10}{10000+t}$
seed number	26

Table A.4.: Learning parameter.

In the 10 digit example with 12x12 pixels in Section 4.3.2 we applied:

learning algorithm	PCD
number of visible units	144
number of hidden units	100
batch size	1
training steps	3000000
learning rate $\eta(t)$	$\frac{100}{1000+0.5t}$
seed number	105

Table A.5.: Learning parameter.

For the 10 digit example with 28x28 pixels in Section 4.3.2 and Section 5.3 we used:

learning algorithm	PCD
number of visible units	784
number of hidden units	100
batch size	10
training steps	100000
learning rate $\eta(t)$	$\frac{10}{1000+t}$
seed number	115

Table A.6.: Learning parameter.

The imbalanced 3 digit example with 12x12 pixels in Section 5.4.1 was obtained with the following learning parameter:

learning algorithm	PCD
number of visible units	144
number of hidden units	10
batch size	3
training steps	400000
learning rate $\eta(t)$	$\frac{10}{3000+t}$
seed number	48

Table A.7.: Learning parameter.

To train the imbalanced 10 digit example with 12x12 pixels in Section 5.4.2 we used the following parameter:

learning algorithm	PCD
number of visible units	144
number of hidden units	100
batch size	10
training steps	500000
learning rate $\eta(t)$	$\frac{100}{1000+t}$
seed number	98

Table A.8.: Learning parameter.

A.2.5. t-Distributed Stochastic Neighbor Embedding

For the t-SNE visualizations we used throughout this thesis:

perplexity	20
number of dimensions for PCA pre-processing D_{pre}	50
initial momentum	0.5
final momentum	0.8
learning rate η	500
maximal number of iterations	1000
early exaggeration	4

Table A.9.: t-SNE parameter. The initial momentum is applied for the first 20 steps, afterwards the final momentum. Early exaggeration is used during the first 100 steps. To diminish the computational cost we apply in a pre-processing step normal PCA to reduce the dimension of the input data to D_{pre}. The reduced data is then used as input for the actual t-SNE algorithm.

A.3. Mixing Image Sequences

Figure A.2.: TSO mixing

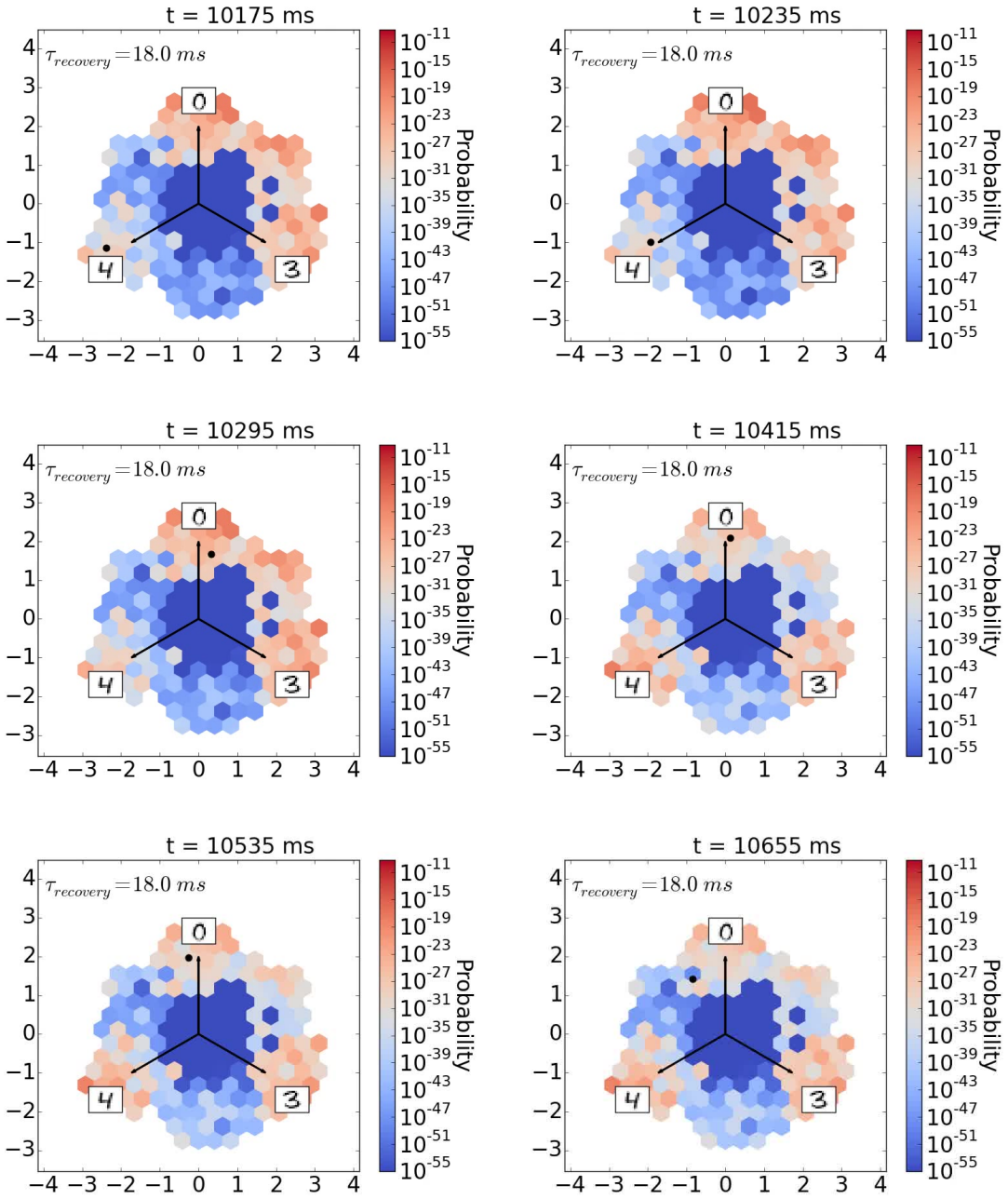


Figure A.2.: TSO mixing

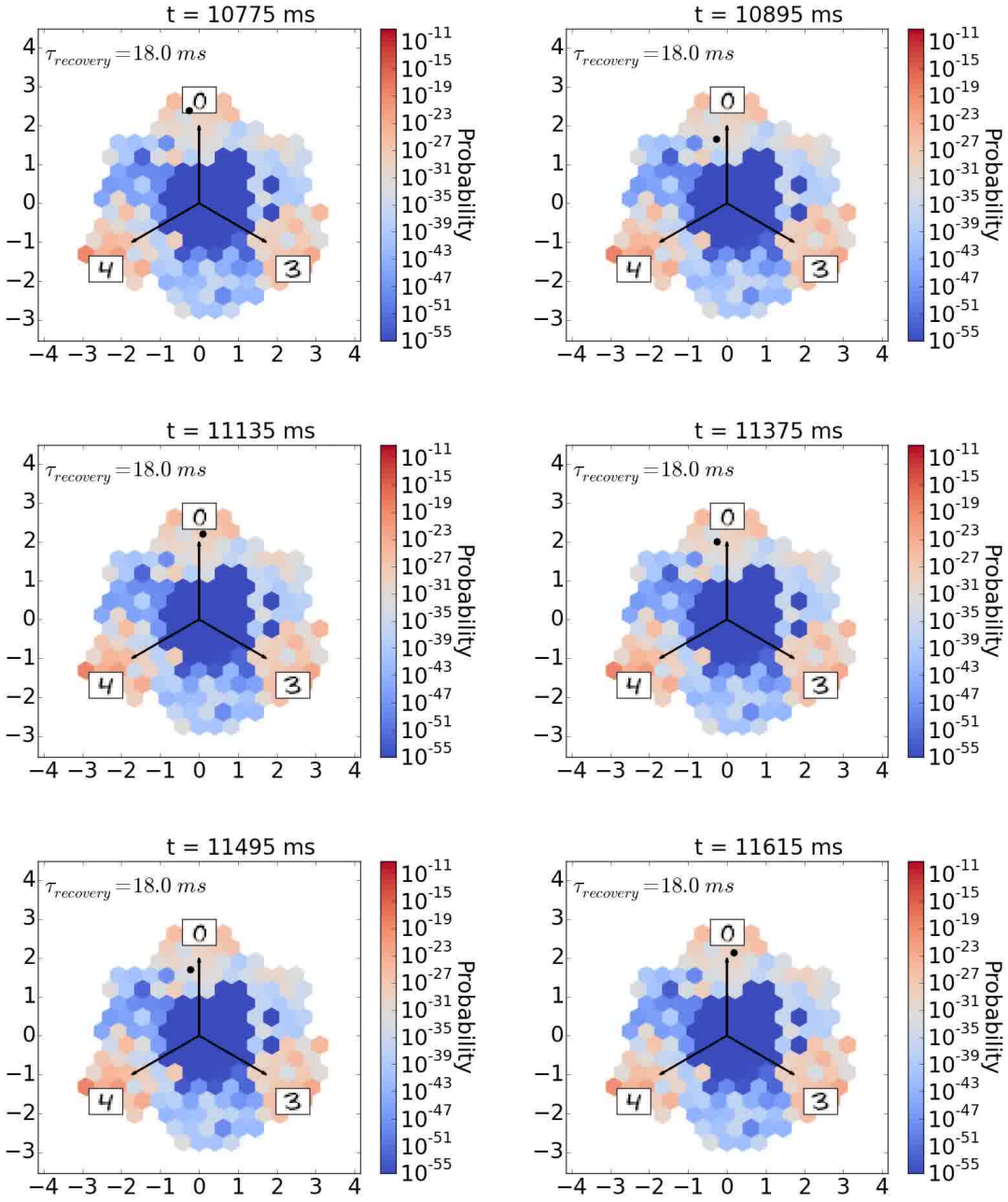


Figure A.2.: TSO mixing

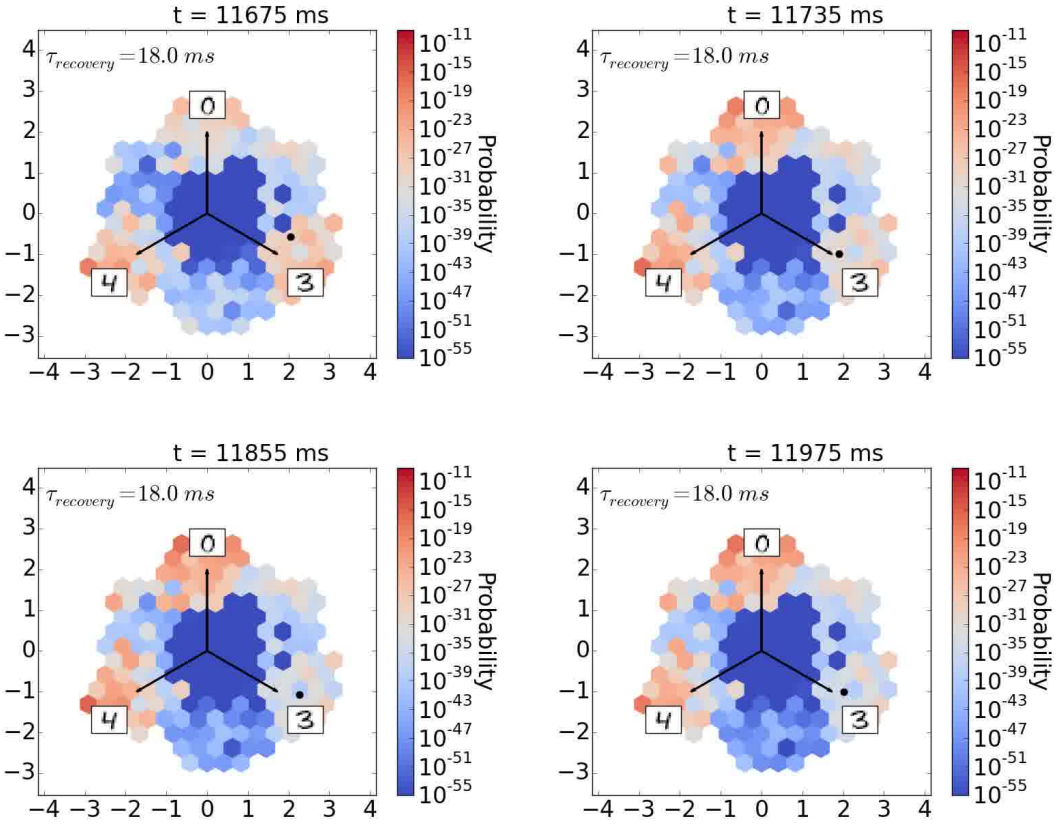


Figure A.3.: Image sequence illustrating the mixing principle of LIF sampling with TSO with a recovery time of $\tau_{\text{rec}} = 18\,\text{ms}$ for exponential-shaped LIF sampling. We used an RBM trained on the 3 digit example like in Section 4.3.1. The basis of this visualization is the star plot (Section 3.10.1) like e.g. Fig. 4.23a. The shown probability landscape is an approximation of the real one, which would be impossible to calculate for 10^{154} possible states. For the approximation we devided the two-dimensional space into hexagons. From a long AST sampling run, where we stored every sample (even the high temperature ones), we obtained samples distributed over the whole area. Afterwards, we assigned to each hexagon one of these samples which is closest to its center. The probability of this sample is used to approximate the probability of the whole hexagon. The image sequence shows an LIF sampling run, where only the current sample is shown as a black dot. The probabilities of the hexagons change for each step because the weights change due to the TSO mechanism. However, as we still use the Boltzmann distribution (3.13) to calculate the probabilities, we make an error because it assumes symmetric weights. Still using (3.13) corresponds to taking the mean of the asymmetric weights for the calculation. This emphasizes that the image sequence is just a rough approximation to the evolution of the probability landscape for illustration purposes.

Figure A.4.: AST mixing

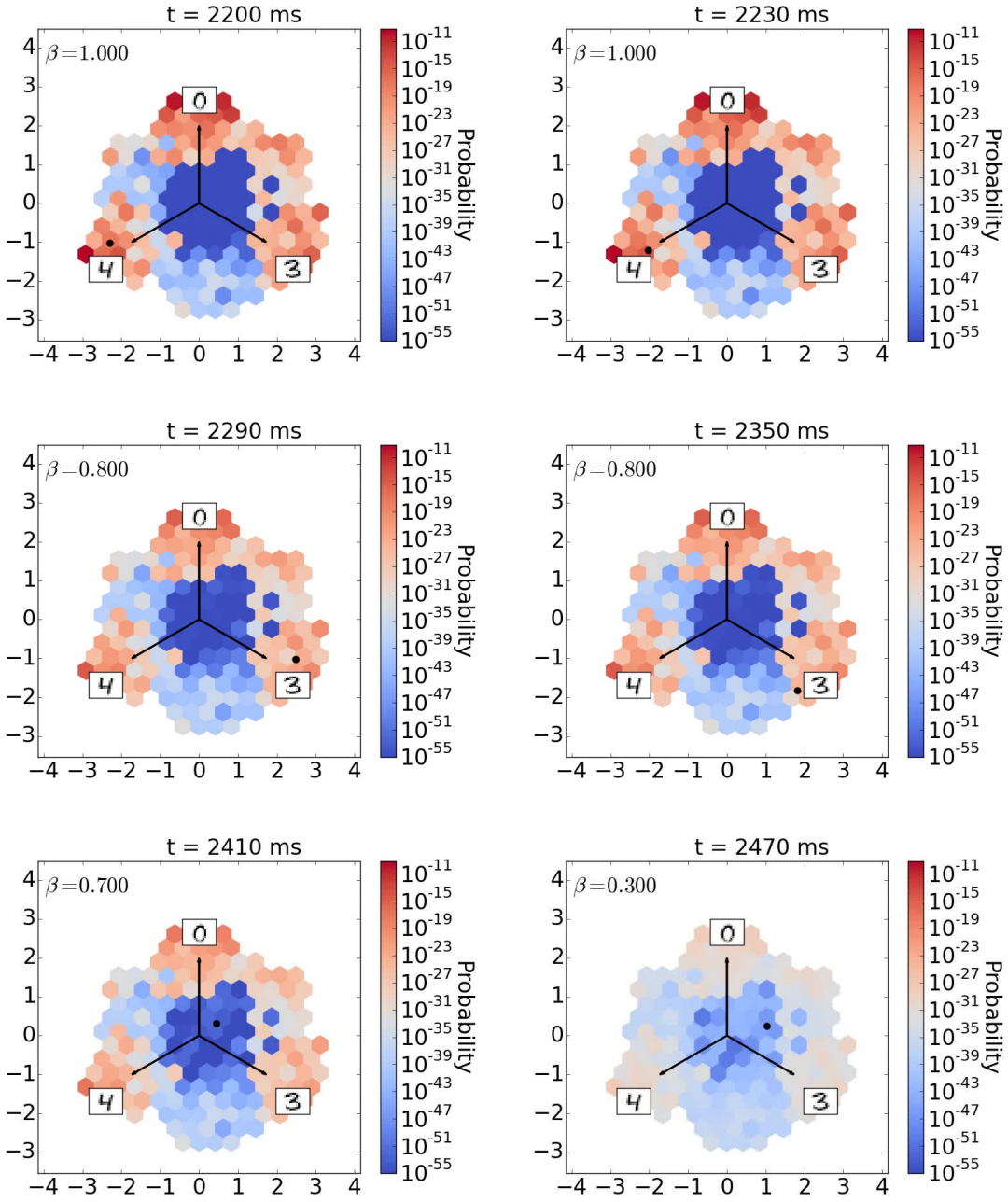
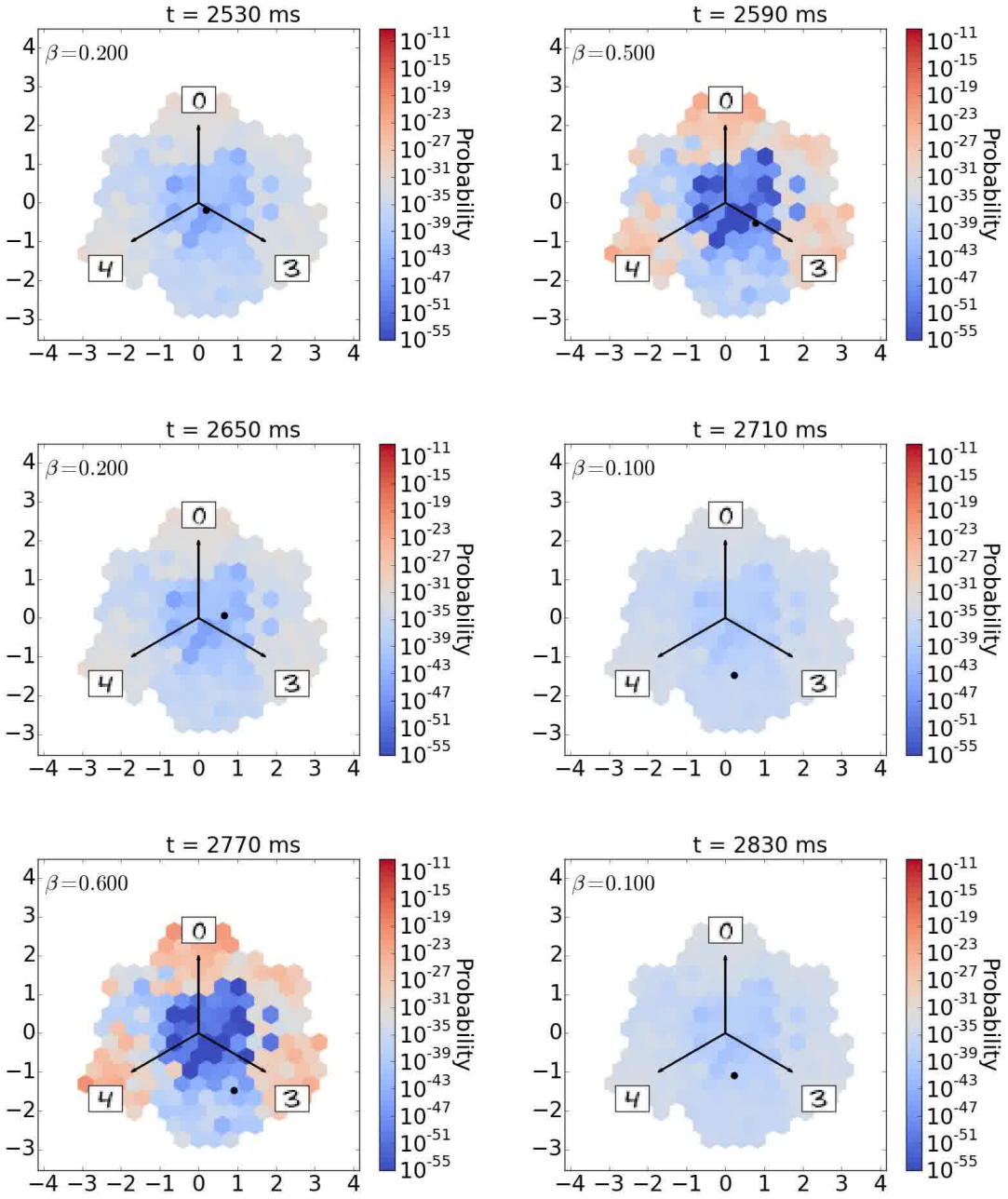


Figure A.4.: AST mixing



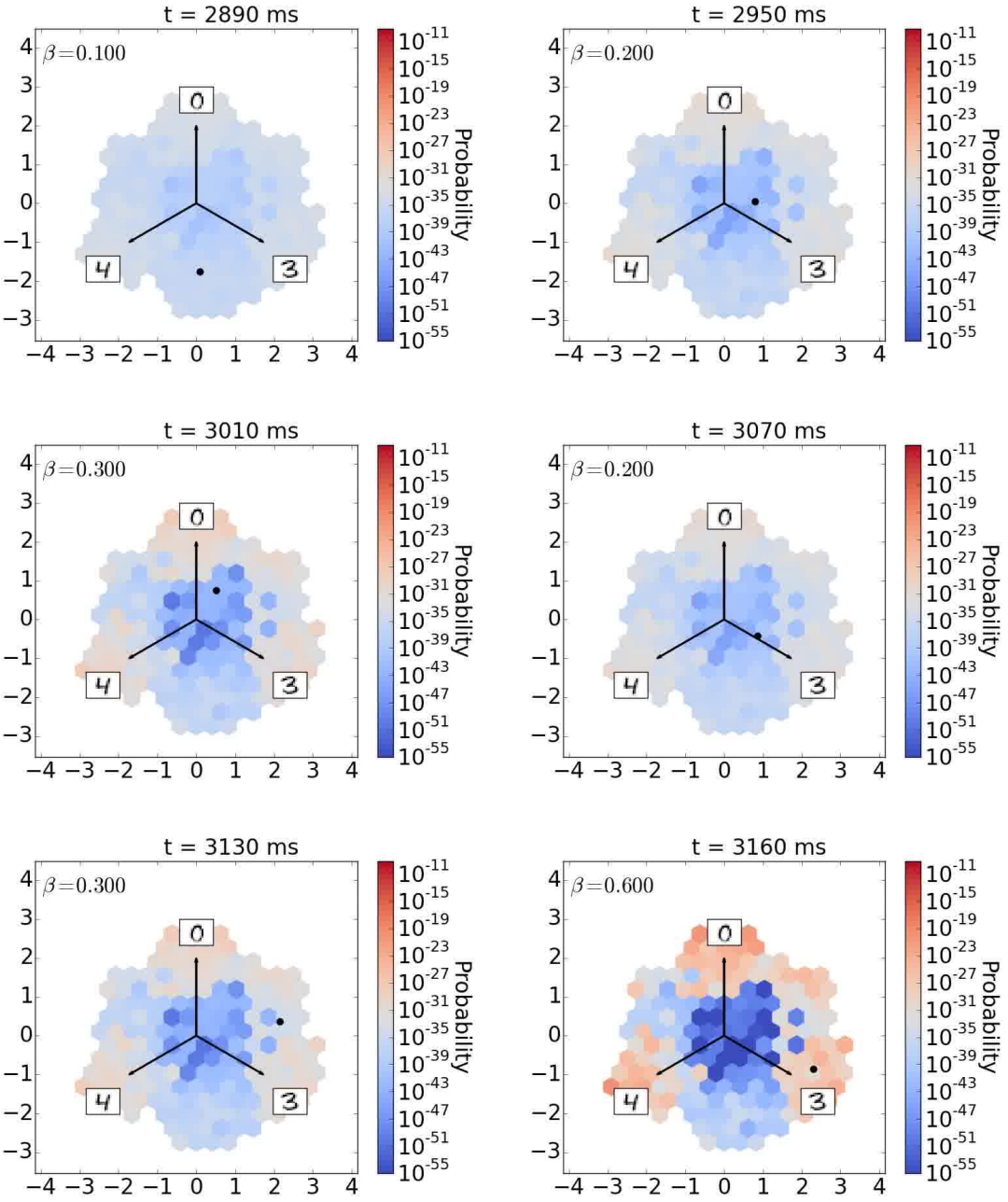


Figure A.4.: Image sequence showing the mixing principle of AST. The same method as in Fig. A.3 was used to create the plot. The only difference is that here the probability distribution is changed due to sampling from different inverse temperature levels according to (3.24). Therefore the current inverse temperature is displayed in each plot.

A.4. Random Homogeneous Distributions - Result Tables

	BM, 5 units		BM, 10 units		BM, 15 units		BM, 20 units	
sampler	DKL mean	DKL error	DKL mean	DKL error	DKL mean	DKL error	DKL mean	DKL error
GS	1.83E-04	3.85E-05	5.10E-03	2.76E-04	1.75E-01	5.64E-03	1.70E+00	1.27E-01
AST	1.59E-04	4.59E-05	5.14E-03	2.58E-04	1.74E-01	4.65E-03	1.70E+00	1.26E-01
curr_exp	1.11E-03	7.12E-04	4.19E-03	9.33E-04	3.79E-02	2.35E-03	4.34E-01	4.32E-02
curr_alpha	6.90E-04	4.59E-04	6.40E-03	2.36E-03	3.31E-02	9.34E-04	4.24E-01	3.84E-02
cond_exp	2.61E-03	1.24E-03	7.68E-03	2.67E-03	4.31E-02	1.71E-03	4.92E-01	4.30E-02
cond_alpha	1.29E-03	4.57E-04	4.43E-03	5.86E-04	3.49E-02	1.24E-03	4.64E-01	3.45E-02

Table A.10.: Final results of the D_{KL} evolutions for normal BMs from Fig. 4.1. We are comparing the results for BMs with 5, 10, 15 and 20 units. The weights and biases have been drawn from a Gaussian distribution with $\mu = 0$ and standard deviation $\sigma = 0.3$. The mean and standard deviation of the D_{KL} values over ten runs are shown where in each run we draw new weights from the Gaussian distribution. "curr/cond" stand for CUBA/COBA, "exp" for exponential-shaped and "alpha" for alpha-shaped synapses.

	BM, 5 units		BM, 10 units		BM, 15 units		BM, 20 units	
sampler	DKL mean	DKL error	DKL mean	DKL error	DKL mean	DKL error	DKL mean	DKL error
GS	1.88E-04	4.12E-05	5.17E-03	2.93E-04	1.79E-01	1.09E-03	1.78E+00	2.94E-03
AST	1.49E-04	4.63E-05	5.21E-03	3.04E-04	1.79E-01	1.24E-03	1.78E+00	3.39E-03
curr_exp	4.58E-04	6.78E-05	4.24E-03	6.31E-05	3.64E-02	4.96E-04	4.41E-01	8.82E-04
curr_alpha	4.49E-04	4.07E-05	3.52E-03	1.17E-04	3.20E-02	3.42E-04	4.29E-01	8.09E-04
cond_exp	1.18E-03	1.53E-04	6.53E-03	2.64E-04	4.43E-02	3.77E-04	5.06E-01	1.54E-03
cond_alpha	1.00E-03	1.07E-04	4.00E-03	2.23E-04	3.50E-02	2.69E-04	4.76E-01	1.20E-03

Table A.11.: Final results of the D_{KL} evolutions for normal BMs, corresponding to the results of Table A.10. The difference is that here we have drawn the weights and biases of the BMs just once. The mean and standard deviation (error) for the D_{KL} values is calculated for 10 runs with different seeds for the random number generators.

	RBM, 5 hidden		RBM, 10 hidden		RBM, 15 hidden		RBM, 20 hidden	
sampler	DKL mean	DKL error	DKL mean	DKL error	DKL mean	DKL error	DKL mean	DKL error
GS	1.78E-04	3.96E-05	4.87E-03	7.17E-04	8.86E-02	2.09E-02	4.18E-01	1.42E-01
AST	1.77E-04	4.53E-05	4.87E-03	6.06E-04	8.84E-02	2.01E-02	4.19E-01	1.42E-01
curr_exp	3.99E-02	1.37E-02	8.03E-02	2.62E-02	1.45E-01	2.92E-02	2.34E-01	4.75E-02
curr_alpha	3.04E-02	1.19E-02	7.10E-02	1.13E-02	1.33E-01	1.58E-02	2.62E-01	4.56E-02
cond_exp	3.26E-02	1.00E-02	6.50E-02	2.02E-02	1.26E-01	2.28E-02	2.25E-01	4.89E-02
cond_alpha	1.39E-02	8.79E-03	3.57E-02	4.89E-03	7.79E-02	1.15E-02	2.14E-01	5.13E-02

Table A.12.: Final results of the D_{KL} evolutions for the RBMs in Section 4.1.1. Displayed are the results for RBMs with 100 visible units and 5, 10, 15 and 20 hidden units. The D_{KL}'s are between the hidden distribution obtained from theory and sampling. The weights and biases have been drawn from a Gaussian distribution with $\mu = 0$ and standard deviation $\sigma = 0.3$. The mean and standard deviation of the D_{KL} values over ten runs are shown, where in each run we draw new weights from the Gaussian distribution. "curr/cond" stand for CUBA/COBA, "exp" for exponential-shaped and "alpha" for alpha-shaped synapses.

	RBM, 5 hidden		RBM, 10 hidden		RBM, 15 hidden		RBM, 20 hidden	
sampler	DKL mean	DKL error	DKL mean	DKL error	DKL mean	DKL error	DKL mean	DKL error
GS	1.87E-04	4.18E-05	5.31E-03	1.40E-04	5.71E-02	6.15E-04	3.62E-01	1.28E-03
AST	1.76E-04	2.31E-05	5.07E-03	2.66E-04	5.75E-02	7.49E-04	3.62E-01	1.62E-03
curr_exp	5.95E-02	1.24E-03	1.11E-01	1.32E-03	1.39E-01	7.72E-04	2.43E-01	1.38E-03
curr_alpha	2.82E-02	8.49E-04	8.60E-02	1.36E-03	1.17E-01	7.45E-04	2.79E-01	2.51E-03
cond_exp	4.57E-02	1.08E-03	9.03E-02	1.12E-03	1.20E-01	1.87E-03	2.29E-01	1.60E-03
cond_alpha	8.55E-03	4.89E-04	3.89E-02	7.07E-04	6.04E-02	1.37E-03	2.09E-01	1.51E-03

Table A.13.: Final results of the D_{KL} evolutions for RBMs. They correspond to the results of Table A.12. The difference is that here we have drawn the weights and biases of the RBMs just once. The mean and standard deviation (error) for the D_{KL} values is calculated for 10 runs with different seeds for the random number generator.

Bibliography

Ackley, D. H., G. E. Hinton, and T. J. Sejnowski, A learning algorithm for Boltzmann machines, *Cognitive Science*, *9*, 147–169, 1985.

Benjamin, B. V., et al., Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations, *Proceedings of the IEEE*, *102*(5), 699–716, 2014.

Billaudelle, S., Characterisation and calibration of short term plasticity on a neuromorphic hardware chip, Bachelor thesis, Ruprecht-Karls-Universität Heidelberg, hD-KIP 14-93, 2014.

Bishop, C. M., *Pattern recognition and machine learning*, vol. 1, springer New York, 2009.

Bishop, C. M., and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 1, springer New York, 2006.

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

BrainScaleS, Research, `http://brainscales.kip.uni-heidelberg.de/public/index.html`, 2012.

Breitwieser, O., Towards a neuromorphic implementation of spike-based expectation maximization, Master thesis, Ruprecht-Karls-Universität Heidelberg, 2015.

Brette, R., and W. Gerstner, Adaptive exponential integrate-and-fire model as an effective description of neuronal activity, *J. Neurophysiol.*, *94*, 3637 – 3642, doi:NA, 2005.

Brette, R., et al., Simulation of networks of spiking neurons: A review of tools and strategies, *Journal of Computational Neuroscience*, *23*(3), 349–398, 2007.

Brüderle, D., et al., A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems, *Biological Cybernetics*, *104*, 263–296, 2011.

Buesing, L., J. Bill, B. Nessler, and W. Maass, Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons, *PLoS Computational Biology*, *7*(11), e1002,211, 2011.

Cattell, R., and A. Parker, Challenges for brain emulation: why is building a brain so difficult, *Natural intelligence*, *1*(3), 2012.

Chawla, N. V., Data mining for imbalanced datasets: An overview, in *Data Mining and Knowledge Discovery Handbook*, pp. 875–886, Springer, 2010.

Davison, A. P., D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, PyNN: a common interface for neuronal network simulators, *Front. Neuroinform.*, *2*(11), 2008.

Dayan, P., and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, The MIT press, Cambride, Massachusetts, 2001.

Diesmann, M., and M.-O. Gewaltig, NEST: An environment for neural systems simulations, in *Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 2001*, *GWDG-Bericht*, vol. 58, edited by T. Plesser and V. Macho, pp. 43–70, Ges. für Wiss. Datenverarbeitung, Göttingen, 2002.

Ehrlich, M., K. Wendt, L. Zühl, R. Schüffny, D. Brüderle, E. Müller, and B. Vogginger, A software framework for mapping neural networks to a wafer-scale neuromorphic hardware system, in *Proceedings of the Artificial Neural Networks and Intelligent Information Processing Conference (ANNIIP) 2010*, pp. 43–52, 2010.

FACETS, Research, `http://http://facets.kip.uni-heidelberg.de/`, 2010.

Fischer, A., and C. Igel, Training restricted boltzmann machines: an introduction, *Pattern Recognition*, *47*(1), 25–39, 2014.

Fuhrmann, G., I. Segev, H. Markram, and M. Tsodyks, Coding of temporal information by activity-dependent synapses, *Journal of neurophysiology*, *87*(1), 140–148, 2002.

Furber, S. B., F. Galluppi, S. Temple, L. Plana, et al., The spinnaker project, *Proceedings of the IEEE*, *102*(5), 652–665, 2014.

Geman, S., and D. Geman, Stochastic relaxation, gibbs distributions, and the bayesian restoration of images, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6), 721–741, 1984.

Gerstner, W., and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, 2002.

Gleeson, P., et al., Neuroml: a language for describing data driven models of neurons and networks with a high degree of biological detail, 2010.

Goodman, D., and R. Brette, Brian: a simulator for spiking neural networks in Python, *Front. Neuroinform.*, *2*(5), 2008.

Hartmann, S., S. Schiefer, S. Scholze, J. Partzsch, C. Mayr, S. Henker, and R. Schuffny, Highly integrated packet-based aer communication infrastructure with 3gevent/s throughput, in *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*, pp. 950–953, doi:10.1109/ICECS.2010.5724670, 2010.

Hastings, W. K., Monte carlo sampling methods using markov chains and their applications, *Biometrika*, *57*(1), 97–109, 1970.

He, H., E. Garcia, et al., Learning from imbalanced data, *Knowledge and Data Engineering, IEEE Transactions on*, *21*(9), 1263–1284, 2009.

Hines, M., and N. Carnevale, *The NEURON simulation environment.*, pp. 769–773, M.A. Arbib, 2003.

Hinton, G., M. Welling, and A. Mnih, Wormholes improve contrastive divergence, *Advances in Neural Information Processing Systems*, *16*, 417–424, 2004.

Hinton, G. E., Training products of experts by minimizing contrastive divergence, *Neural computation*, *14*(8), 1771–1800, 2002.

Hinton, G. E., Boltzmann machine, *2*(5), 1668, revision 91075, 2007.

Hinton, G. E., and S. T. Roweis, Stochastic neighbor embedding, in *Advances in neural information processing systems*, pp. 833–840, 2002.

Hinton, G. E., and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, *Science*, *313*(5786), 504–507, 2006.

Hotelling, H., Analysis of a complex of statistical variables into principal components., *Journal of educational psychology*, *24*(6), 417, 1933.

Hunter, J. D., Matplotlib: A 2D graphics environment, *IEEE Computing in Science and Engineering*, *9*(3), 90–95, 2007.

Inc., A. S., Postscript language reference manual, 1999.

Javed, F., et al., Brain and high metabolic rate organ mass: contributions to resting energy expenditure beyond fat-free mass, *The American journal of clinical nutrition*, *91*(4), 907–912, 2010.

Jolliffe, I., *Principal component analysis*, Wiley Online Library, 2002.

Jordan, J., et al., Neural networks as sources of uncorrelated noise for functional neural systems, *Tech. rep.*, Computational and Systems Neuroscience, 2014.

Kubat, M., S. Matwin, et al., Addressing the curse of imbalanced training sets: one-sided selection, in *ICML*, vol. 97, pp. 179–186, Nashville, USA, 1997.

Lapicque, L., Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization, *Journal de Physiologie et Pathologie General*, *9*, 620–635, 1907.

Larochelle, H., and Y. Bengio, Classification using discriminative restricted boltzmann machines, in *Proceedings of the 25th international conference on Machine learning*, pp. 536–543, ACM, 2008.

Le Roux, N., N. Heess, J. Shotton, and J. Winn, Learning a generative model of images by factoring appearance and shape, *Neural Computation*, *23*(3), 593–650, 2011.

LeCun, Y., and C. Cortes, The mnist database of handwritten digits, 1998.

Leng, L., Deep learning architectures for neuromorphic hardware, Master thesis, Ruprecht-Karls-Universität Heidelberg, hD-KIP 14-26, 2014.

Lichtsteiner, P., C. Posch, and T. Delbruck, A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor, *Solid-State Circuits, IEEE Journal of*, *43*(2), 566–576, 2008.

Liu, S.-C., and T. Delbruck, Neuromorphic sensory systems, *Current opinion in neurobiology*, *20*(3), 288–295, 2010.

Maass, W., and H. Markram, Synapses as dynamic memory buffers, *Neural Networks*, *15*(2), 155–161, 2002.

Marinari, E., and G. Parisi, Simulated tempering: a new monte carlo scheme, *EPL (Europhysics Letters)*, *19*(6), 451, 1992.

Markram, H., The human brain project, *Scientific American*, *306*(6), 50–55, 2012.

Markram, H., A. Gupta, A. Uziel, Y. Wang, and M. Tsodyks, Information processing with frequency-dependent synaptic connections., *Neurobiol Learn Mem*, *70*(1-2), 101–112, 1998.

Mead, C. A., *Analog VLSI and Neural Systems*, Addison Wesley, Reading, MA, 1989.

Mead, C. A., Neuromorphic electronic systems, *Proceedings of the IEEE*, *78*, 1629–1636, 1990.

Mead, C. A., and M. A. Mahowald, A silicon model of early visual processing, *Neural Networks*, *1*(1), 91–97, 1988.

Merolla, P. A., et al., A million spiking-neuron integrated circuit with a scalable communication network and interface, *Science*, *345*(6197), 668–673, 2014.

Metropolis, N., and S. Ulam, The monte carlo method, *Journal of the American statistical association*, *44*(247), 335–341, 1949.

Mohamed, A.-r., and G. Hinton, Phone recognition using restricted boltzmann machines, in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 4354–4357, IEEE, 2010.

Morrison, A., C. Mehring, T. Geisel, A. Aertsen, and M. Diesmann, Advancing the boundaries of high-connectivity network simulation with distributed computing, *Neural computation*, *17*(8), 1776–1801, 2005.

Naud, R., N. Marcille, C. Clopath, and W. Gerstner, Firing patterns in the adaptive exponential integrate-and-fire model, *Biological cybernetics*, *99*(4-5), 335–347, 2008.

Neil, D., and S.-C. Liu, Minitaur, an event-driven fpga-based spiking network accelerator, *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, *22*(12), 2621–2628, 2014.

Numpy, Website, `http://numpy.scipy.org`, 2012.

Pecevski, D. A., T. Natschläger, and K. N. Schuch, Pcsim: A parallel simulation environment for neural circuits fully integrated with Python, *Front. Neuroinform.*, *3*(11), 2009.

Petrovici, M. A., Function vs. substrate: Theory and models for neuromorphic hardware, Ph.D. thesis, 2015.

Petrovici, M. A., J. Bill, I. Bytschok, J. Schemmel, and K. Meier, Stochastic inference with deterministic spiking neurons, *arXiv preprint arXiv:1311.3211*, 2013.

Probst, D., A neural implementation of probabilistic inference in binary probability spaces, Master thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

Probst, D., M. A. Petrovici, I. Bytschok, J. Bill, D. Pecevski, J. Schemmel, and K. Meier, Probabilistic inference in discrete spaces can be implemented into networks of lif neurons, *Frontiers in computational neuroscience*, *9*, 2015.

Ray, S., and U. S. Bhalla, PyMOOSE: interoperable scripting in Python for MOOSE, *Front. Neuroinform.*, *2*(6), 2008.

Roth, M., Predictive stochastic inference - from abstract models to neuromorphic implementation, Bachelor thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

Salakhutdinov, R., Learning deep boltzmann machines using adaptive mcmc, in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 943–950, 2010.

Salakhutdinov, R., and G. E. Hinton, Deep boltzmann machines, in *International Conference on Artificial Intelligence and Statistics*, pp. 448–455, 2009.

Salakhutdinov, R., A. Mnih, and G. Hinton, Restricted boltzmann machines for collaborative filtering, in *Proceedings of the 24th international conference on Machine learning*, pp. 791–798, ACM, 2007.

Salakhutdinov, R. R., Learning in markov random fields using tempered transitions, in *Advances in neural information processing systems*, pp. 1598–1606, 2009.

Schemmel, J., A. Grübl, K. Meier, and E. Muller, Implementing synaptic plasticity in a VLSI spiking neural network model, in *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN)*, IEEE Press, 2006.

Schemmel, J., D. Brüderle, K. Meier, and B. Ostendorf, Modeling synaptic plasticity within networks of highly accelerated I&F neurons, in *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 3367–3370, IEEE Press, 2007.

Schemmel, J., J. Fieres, and K. Meier, Wafer-scale integration of analog neural networks, in *Proceedings of the 2008 International Joint Conference on Neural Networks (IJCNN)*, 2008.

Schemmel, J., D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, A wafer-scale neuromorphic hardware system for large-scale neural modeling, in *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1947–1950, 2010.

Schmah, T., G. E. Hinton, S. L. Small, S. Strother, and R. S. Zemel, Generative versus discriminative training of rbms for classification of fmri images, in *Advances in neural information processing systems*, pp. 1409–1416, 2008.

Smolensky, P., Information processing in dynamical systems: Foundations of harmony theory, 1986.

Tang, Y., R. Salakhutdinov, and G. Hinton, Robust boltzmann machines for recognition and denoising, in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 2264–2271, IEEE, 2012.

Taylor, G. W., and G. E. Hinton, Factored conditional restricted boltzmann machines for modeling motion style, in *Proceedings of the 26th annual international conference on machine learning*, pp. 1025–1032, ACM, 2009.

Taylor, G. W., G. E. Hinton, and S. T. Roweis, Modeling human motion using binary latent variables, in *Advances in neural information processing systems*, pp. 1345–1352, 2006.

Tieleman, T., Training restricted boltzmann machines using approximations to the likelihood gradient, in *Proceedings of the 25th international conference on Machine learning*, pp. 1064–1071, ACM, 2008.

Tieleman, T., and G. Hinton, Using fast weights to improve persistent contrastive divergence, in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1033–1040, ACM, 2009.

Tsodyks, M., and H. Markram, The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability, *Proceedings of the national academy of science USA*, *94*, 719–723, 1997.

Van der Maaten, L., and G. Hinton, Visualizing data using t-sne, *Journal of Machine Learning Research*, *9*(2579-2605), 85, 2008.

Vogelstein, R. J., U. Mallik, E. Culurciello, and R. E.-C. Gert Cauwenberghs, A multichip neuromorphic system for spike-based visual information processing, *Neural Computation*, *19*, 2281–2300, 2007.

Wang, F., and D. P. Landau, Efficient, multiple-range random walk algorithm to calculate the density of states, *Physical Review Letters*, *86*(10), 2050, 2001.

Weilbach, C., An online learning algorithm for lif-based boltzmann machines, Bachelor thesis, Ruprecht-Karls-Universität Heidelberg, 2015.

Welling, M., A. Mnih, and G. E. Hinton, Wormholes improve contrastive divergence, in *Advances in Neural Information Processing Systems*, p. None, 2003.

Yu, T., and G. Cauwenberghs, Analog vlsi neuromorphic network with programmable membrane channel kinetics, in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pp. 349–352, IEEE, 2009.

Acknowledgments (Danksagungen)

Ich danke:

Prof. Dr. Karlheinz Meier und Dr. Johannes Schemmel für die Aufnahme in der Electronic Vision(s) Gruppe und die Möglichkeit diese Arbeit zu schreiben.

Mihai für die intensive Betreuung und viele lehrreiche Gespräche.

Christian, Ilja, Oliver, Vitali und Luziwei für die einzigartige Atmosphäre im Büro.

Allen Modelern für viele interessante Diskussionen.

Der gesamten Vision(s) Gruppe für das Beantworten vieler Fragen, den Diskussionen beim Mittagessen und der besonderen Gruppenatmosphäre.

Oliver, Vitali, Mihai, Luziwei und beiden Christians für das Korrekturlesen dieser Arbeit.

Meiner Familie für die Unterstützung all die Jahre, die mir das Studium erst ermöglicht hat.

Statement of Originality (Erklärung):

I certify that this thesis, and the research to which it refers, are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, November 25, 2015

.....
(signature)