RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

冷卢子未 Luziwei Leng

Deep Learning Architectures for Neuromorphic Hardware

Master's Thesis

KIRCHHOFF-INSTITUT FÜR PHYSIK

Faculty of Physics and Astronomy
University of Heidelberg

Master's thesis

in Physics

submitted by

冷卢子未 Luziwei Leng

born in Yifeng, Jiangxi, China

Deep Learning Architectures for Neuromorphic Hardware

This master's thesis has been carried out by 冷卢子未 Luziwei Leng at the

Kirchhoff Institute for Physics

Ruprecht-Karls-Universitt Heidelberg

under the supervision of

Prof. Dr. Karlheinz Meier

Deep Learning Architectures for Neuromorphic Hardware

Networks that perform stochastic inference lie at the intersection of neuroscience and machine learning. For the former, they hold the potential of explaining how inference is performed by the mammalian neocortex. For the latter, they offer a very powerful solution to hard problems such as image recognition and classification. During the past several years, theories have been developed that connect the dynamics of biological neural networks to those of the so-called Boltzmann machines that are widely used in machine learning. Based on the theory developed in *Petrovici et al.* (2013), we demonstrate the implementation of deep Boltzmann machines with leaky integrate-and-fire (LIF) neurons. Beyond simply translating parameters between classical and neural Boltzmann machines, we also implement a combination of several powerful learning algorithms and demonstrate its efficiency by training LIF-based Boltzmann machines as a generative model of a set of handwritten digits from the MNIST database. In building these learning neural networks, we identify and discuss several effects that are unique to our choice of neural implementation. We propose that, with proper modifications to our suggested algorithms, our architectures can be embedded in accelerated neuromorphic hardware, thereby fostering the development of powerful, fast and low-power learning machines.

Lernfhige Netzwerke f ü r neuromorphe Hardware

Netzwerke, die stochastische Inferenz ausf ü hren, liegen im bergangsbereich zwischen Neurowissenschaft und maschinellem Lernen. Aus neurowissenschaftlicher Perspektive sind diese Architekturen gute Kandidaten, um Inferenzprozesse im Neokortex zu erklren. Aus der Perspektive k ü nstlicher Intelligenz bieten diese Modelle eine effiziente Lsung f ü r schwierige Probleme, wie zum Beispiel Mustererkennung und -klassifizierung. In den vergangenen Jahren wurden Theorien entwickelt, die przise Analogien zwischen der Dynamik von biologischen neuronalen Netzwerken und der von, im maschinellen Lernen weit verbreiteten, sogenannten Boltzmann-Maschinen ermglichen. Ausgehend von der Theorie aus *Petrovici et al.* (2013), entwickeln wir eine Implementierung von mehrschichtigen Boltzmann-Maschinen mit "leaky integrate-and-fire" (LIF) Neuronen. Zustzlich zur direkten bersetzung der Parameter zwischen klassischen und neuronalen Boltzmann-Maschinen, implementieren wir eine Kombination verschiedener mchtiger Lernalgorithmen und demonstrieren ihre Effizienz durch Trainieren vom Boltzmann-Maschinen als generatives Modell zur Reprsentation einer Auswahl handgeschriebener Ziffern aus der MNIST-Datenbank. Bei der Beschreibung dieser lernenden neuronalen Netzwerke werden Effekte identifiziert und diskutiert, welche f ü r unsere Wahl einer neuronalen Implementierung einzigartig zu sein scheinen. Unsere Ergebnisse legen nahe, dass durch gezielte Modifikationen der vorgestellten Lernalgorithmen, sich unsere Netzwerke auf beschleunigter neuromorpher Hardware implementieren lassen, was die Entwicklung mchtiger, schneller und energieeffizienter lernender Maschinen erlauben w ü rde.

Contents

1. Introduction

Today, the human brain remains a scientific puzzle. Constant efforts during the past century, coming from multiple disciplinary fields, have expanded the knowledge base of neuroscience enormously. However, most of this knowledge is either markedly microscopic (concerning the biochemistry and electrophysiology of individual neurons and synapses) or very high-level macroscopic (such as mm-resolution FMRI data). The intermediary, network level, which arguably holds the key to understanding computation in the brain, is still largely unknown.

Owing to the technological developments of the computer era, the "brain problem" can be attacked from a different angle. By formulating hypotheses about the structure of neural networks, computational neuroscientists can simulate their model networks at various spatial-temporal levels to produce predictions that can then be validated by biological data. Furthermore, even if they do not accurately describe structures found within the brain, such biologically inspired neural networks have found various applications in the fields of robotics, machine learning and AI.

The term "computational neuroscience" was introduced by Eric L. Schwartz in 1985 However, research related to this field can be traced back to as far as the early 20th century. In 1907, Lapicque introduced the integrate and fire neuron model, which is still one of the most popular models in computational neuroscience. Despite its simplicity – which is of great advantage from both a theoretical and a computational point of view – it offers a remarkably good approximation of neural membrane dynamics. About 40 years later, Hodgkin & Huxley created the first biophysical model of the action potential, and described it with a set of nonlinear differential equations. However, many of today's network models are based essentially on Lapicque's neuron, augmented by a simple threshold for triggering an action potential, which is modeled simply as a δ-function.

1.1. Neural networks, probabilistic inference and sampling

It is, in some way, a truism, to say that the brain performs, on some level, probabilistic inference. After all, each conscious movement is the result of processed sensory data, combined with an expectation of the future state of the body within the environment. However, inference is likely to play a role in many more areas besides the sensorimotor apparatus. The inference hypothesis has, indeed gained support by recent experiments, such as, e.g., *Knill and Pouget* (2004). Additionally, *Fiser et al.* (2010) has suggested a sampling-based framework for inference in the cortex of the human brain.

In 2011, the so-called neural sampling theory was proposed (*Buesing et al.*, 2011), which took an essential step towards understanding the microscopic implementation of sampling-based probabilistic inference. The authors provided a mathematical framework in which the stochastic firing activity of a network based on abstract model neurons can be interpreted as Markov chain Monte Carlo sampling. However, a certain gap towards biology remained: the largely deterministic nature of biological neuron membrane dynamics stands somewhat in contrast to their inherently stochastic, abstract neuron model.

Building on the work from *Buesing et al.* (2011), *Petrovici et al.* (2013) refined the neural sampling theory and extended it to a more biologically plausible neuron model. Their work demonstrated that in a spiking noisy environment, leaky integrate-and-fire (LIF) neurons can indeed perform probabilistic inference through sampling from well-defined probability distributions. In particular, they have shown that such neural networks can emulate Boltzmann machines, thereby creating a rigorous link between this powerful machine-learning model and the dynamics of biological spiking neurons.

1.2. Exploring the potential of sampling neurons: LIF-based Boltzmann machines

Proposed by Hinton and Sejnowski in 1985, a Boltzmann machine is a type of recurrent network consisting of stochastic binary units. With certain constrains on its connectivity, a general Boltzmann machine becomes a so-called "restricted" Boltzmann machine, which allows the implementation of efficient learning algorithms, thereby becoming useful for many practical problems. Recently, RBMs were proposed in *Hinton et al.* (2006) as building blocks of multi-layer learning networks called deep belief networks, and were used in *Salakhutdinov and Hinton* (2009) to construct so-called "deep" Boltzmann machines. Studies such as *Srivastava and Salakhutdinov* (2012) showed that deep Boltzmann machines combined with state-of-art learning algorithms can outperform many other often-used machine learning models, such as support vector machines and linear discriminant analysis, in various classification tasks.

Petrovici et al. (2013) have already shown that networks of LIF neurons can accurately sample from Boltzmann distributions. To further explore their potential for hard computational tasks, Boltzmann machines with more complex target distributions need to be studied.

Throughout this work, we investigate the combination of the LIF-based Boltzmann machines with efficient learning algorithms. We show that powerful machine learning algorithms are, indeed, applicable to networks of LIF neurons. In demonstrating the performance of this approach, we follow a twofold long-term agenda. As a more immediate application, we envision the emulation of such networks on neuromorphic hardware, which offers significant advantages over conventional computing architectures in terms of power consumption and speed. On the long run, we hope that our study of these models

will facilitate biological investigations of similar learning and sampling mechanisms in the mammalian neocortex.

1.3. Outline

In Chapter 2, we discuss general Boltzmann machines and learning rules proposed in recent literature. After a brief review of Gibbs sampling, we discuss the neural sampling theory from *Buesing et al.* (2011) for neurons with an absolute refractory mechanism. In the end, we sketch the theory of sampling with LIF neurons from *Petrovici et al.* (2013), which represents the framework for the LIF-based Boltzmann machines throughout the rest of this work.

Chapter 3 focuses on the application of LIF-based Boltzmann machines to model perceptual multistability. The performance of the LIF implementation is compared directly to the implementation with abstract neurons proposed by *Buesing et al.* (2011). In this section, we also explicitly discuss learning algorithms for LIF-based fully visible Boltzmann machines.

In chapter 4 we explore the implementation of RBMs with LIF neurons. RBMs and a series of corresponding learning algorithms are discussed. These learning algorithms are then be combined to form an efficient learning algorithm: coupled adaptive simulated tempering (CAST). Based on this algorithm, both abstract and LIF-based RBMs are trained to learn a set of images of handwritten digits. In the end, our recent progress on deep Boltzmann machines is presented in brief.

Chapter 5 will give a very brief report on the early investigation of the applicability of LIF sampling framework to the currently available BrainScaleS neuromorphic hardware. We then conclude this work with a summary and outlook in Chapter 6.

2. LIF-based Boltzmann machines: theoretical background

In the previous chapter, we suggested that biological neurons can perform stochastic inference and can therefore be used to implement state-of-the-art machine learning algorithms. As we are interested in ensembles that can represent probabilistic spaces and perform inference therein, we will first review several fundamental concepts of stochastic computing. We will start with a short discussion of sampling methods, in particular Gibbs sampling. Afterwards, we will focus on a particular class of probabilistic distribution, namely Boltzmann distributions, and their physical implementation in so-called Boltzmann machines. The latter can be realized with abstract model neurons within the concept of so-called neural sampling. This model can be further refined to include biologically plausible neuron and synapse models in a framework called LIF sampling. This final embodiment of Boltzmann distributions will constitute the basis for all further studies within this manuscript. Throughout this work, we shall use neural sampling as a theoretically optimal benchmark against which LIF sampling can be characterized.

2.1. Gibbs sampling

Gibbs sampling (GS) is a widely used Markov chain Monte Carlo (MCMC) sampling method. Named after Andrey Markov, a Markov chain is a mathematical system that undergoes transitions from one state to another in a well-defined state space. It is a limited-memory stochastic process: the next state depends only on the a fixed number of preceding states. This memoryless feature is called the Markov property. A first-order Markov chain is defined to be a series of random variables $\mathbf{z}^{(1)}, ..., \mathbf{z}^{(N)}$ such that, at any time, the state of the system only depends on its last state:

$$p(\mathbf{z}^{(k+1)}|\mathbf{z}^{(1)}, ..., \mathbf{z}^{(k)}) = p(\mathbf{z}^{(k+1)}|\mathbf{z}^{(k)}) \,, \tag{2.1}$$

where $k \in \{1, ..., N-1\}$.

Consider the distribution $p(\mathbf{z}) = p(z_1, ..., z_M)$ from which we wish to sample, and suppose we start from some state in the Markov chain. Each step of GS involves replacing the value of one of the variables by a value drawn from the distribution of that variable conditioned on the values of the remaining variables.

For example, suppose we have a distribution $p(z_1, z_2, z_3)$, and at step k of GS we have the current state $z_1^{(k)}, z_2^{(k)}$ and $z_3^{(k)}$. The algorithm first replaces $z_1^{(k)}$ by a new value

$z_1^{(k+1)}$ obtained by sampling from the conditional distribution

$$p(z_1|z_2^{(k)}, z_3^{(k)}) \ . \tag{2.2}$$

Next, it replaces $z_2^{(k)}$ by a value $z_2^{(k+1)}$ obtained by sampling from the conditional distribution

$$p(z_2|z_1^{(k+1)}, z_3^{(k)}) \tag{2.3}$$

so that the new value for z_1 is used directly in subsequent sampling steps. Then it updates z_3 with a sample $z_3^{(k+1)}$ drawn from

$$p(z_3|z_1^{(k+1)}, z_2^{(k+1)}) \tag{2.4}$$

and so on, cycling repeatedly through the three variables in this well-defined order.

2.2. Boltzmann machines

A Boltzmann machine (BM) is a type of recurrent neural network which consists of symmetrically connected stochastic binary units. The energy of the state $\{\mathbf{z}\}$ is defined as

$$E(\mathbf{z}) = -\sum_{i<j} W_{ij} z_i z_j - \sum_i b_i z_i \ , \tag{2.5}$$

where z_i and z_j are the binary states of units i and j, W_{ij} is the weight connecting units z_i and z_j and b_i is the bias of unit z_i. The weights satisfy $W_{ii} = 0$ and $W_{ij} = W_{ji}$ (i.e. the weight matrix is symmetrical and zero-diagonal). The network assigns a probability to a state vector via an energy function $E(\mathbf{z})$

$$p(\mathbf{z}) = \frac{1}{Z} \exp[-E(\mathbf{z})] \ , \tag{2.6}$$

where $Z = \sum_{\mathbf{z}} \exp[-E(\mathbf{z})]$ represents the so-called partition function. The units can be further subdivided into a set of visible units $\mathbf{v}$, whose states can be determined by training data (eg. pixels of a series of images), and a set of hidden units $\mathbf{h}$, whose states can not be directly determined by training data and act as latent variables (see figure 2.1). The energy of the state $\{\mathbf{v}, \mathbf{h}\}$ is then defined as

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{i,j} W_{ij} v_i h_j - \sum_{i<i'} L_{ii'} v_i v_{i'} - \sum_{j<j'} J_{jj'} h_j h_{j'} \ , \tag{2.7}$$

where v_i and h_j are the binary states of visible unit i and hidden unit j and a_i and b_j are their biases, W_{ij}, $L_{ii'}$ and $J_{jj'}$ represent the visible-to-hidden, visible-to-visible and hidden-to-hidden connection weights (also see Fig. 2.1 comments). For a BM with

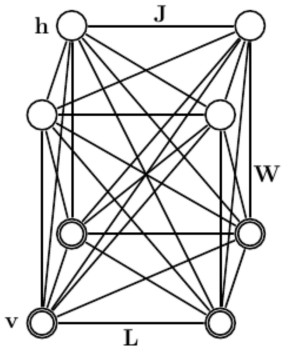


Figure 2.1.: A Boltzmann machine with visible units **v** and hidden units **h**. **W**, **L** and **J** are symmetric, zero-diagnal matrices that contain the visible-to-hidden, visible-to-visible and hidden-to-hidden couplings (Image is taken from *Salakhutdinov and Hinton*, 2009).

visible and hidden units, its output is often related with the visible units' states. The probability for a particular state of the visible units to occur in a BM is given by summing (marginalizing) over all possible hidden vectors

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} \exp\left[-E(\mathbf{v}, \mathbf{h})\right] \ , \tag{2.8}$$

2.2.1. Learning

Apart from being able to represent well-defined Boltzmann distributions, BMs can also be trained to sample from particular areas of the state space with high probability. In other words, given a set of training samples, a BM is able to learn to generate similar states with high probability. Since the stochastic dynamics of a BM favors state vectors that have low energy values, during the learning process, its parameters are updated to lower the energy function of the data vectors in the training set.

By differentiating Eq. 2.6 and using the fact $\partial E(\mathbf{z})/\partial W_{ij} = -z_i z_j$, it can be shown that

$$\begin{aligned}
\frac{\partial p(\mathbf{z})}{\partial W_{ij}} &= \frac{\partial}{\partial W_{ij}} \left[\frac{e^{-E(\mathbf{z})}}{\sum_{\mathbf{z}'} e^{-E(\mathbf{z}')}} \right] \\
&= e^{-E(\mathbf{z})} \cdot z_i \cdot z_j \cdot \left[\sum_{\mathbf{z}'} e^{-E(\mathbf{z}')}\right]^{-1} - e^{-E(\mathbf{z})} \cdot \left[\sum_{\mathbf{z}'} e^{-E(\mathbf{z}')}\right]^{-2} \cdot \sum_{\mathbf{z}'} \left[e^{-E(\mathbf{z}')} \cdot z_i' \cdot z_j'\right] \\
&= p(\mathbf{z}) \cdot z_i \cdot z_j - p(\mathbf{z}) \left\{ \frac{\sum_{\mathbf{z}'} \left[e^{-E(\mathbf{z}')} \cdot z_i' \cdot z_j'\right]}{\sum_{\mathbf{z}'} e^{-E(\mathbf{z}')}} \right\} \ ,
\end{aligned} \tag{2.9}$$

where $\sum_{\mathbf{z}'}$ is a sum over all possible states of the model. Moving the $p(\mathbf{z})$ from the lhs. of the equation to the rhs., we get

$$\frac{\partial \log p(\mathbf{z})}{\partial W_{ij}} = z_i \cdot z_j - \left\{ \frac{\sum_{\mathbf{z}'} \left[e^{-E(\mathbf{z}')} \cdot z_i' \cdot z_j' \right]}{\sum_{\mathbf{z}'} e^{-E(\mathbf{z}')}} \right\} , \tag{2.10}$$

which can be further transformed to

$$\left\langle \frac{\partial \log p(\mathbf{z})}{\partial W_{ij}} \right\rangle_{\text{data}} = \langle z_i \cdot z_j \rangle_{\text{data}} - \langle z_i z_j \rangle_{\text{model}} , \tag{2.11}$$

where $\langle \cdot \rangle_{\text{data}}$ denotes an average over all the training samples and $\langle \cdot \rangle_{\text{model}}$ denotes the expectation value of the distribution defined by the model. This can lead to a learning rule (*Hinton*, 2010) for performing gradient ascent in the log-probability of the training data

$$\Delta W_{ij} = \eta(\langle z_i z_j \rangle_{\text{data}} - \langle z_i z_j \rangle_{\text{model}}) , \tag{2.12}$$

where η represents a learning rate. The learning rule for the bias b_i is the same as in Eq. 2.11, but with z_j omitted:

$$\Delta b_i = \eta(\langle z_i \rangle_{\text{data}} - \langle z_i \rangle_{\text{model}}) . \tag{2.13}$$

However, the computation of $\langle z_i z_j \rangle_{\text{model}}$ requires the calculation of the partition function of the model, which becomes exponentially expensive as the number of units increases. As an alternative, an approximation of $\langle z_i z_j \rangle_{\text{model}}$ can be made by drawing an appropriate sample from the model distribution. Proposed by Hinton (*Hinton*, 2002), contrastive divergence (CD) approximates the expectation value for the model distribution by initializing the model with a training vector $\mathbf{z}^{(0)}$ from the training set and collecting a number of samples after allowing the BM to freely evolve for k steps of Gibbs sampling. Although it can be shown that CD actually is not following the gradient in Eq. 2.11 (*Sutskever and Tieleman*, 2010), it has been proven to work well enough in many applications (*Hinton*, 2010).

Since the training samples do not give any information about the states of hidden units, the learning algorithms for multilayer BMs are more complicated and will be introduced in chapter 4.

2.3. Neural sampling

Recently, a theory (*Buesing et al.*, 2011) has been suggested, which implements MCMC sampling in networks of abstract model neurons (AMNs). This made it possible to build a BM based on spiking neurons.

In AMNs, a binary vector $(z_1, ..., z_K)$ is represented by the firing activity of the

network at time t as follows:

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

which means that any spike of neuron v_k sets the value of the associated binary variable z_k to 1 for a duration of length τ.

For the construction of the sampling network, a so-called neural computability condition (NCC) is assumed, which defines the membrane potential $u_k(t)$ of neuron k at time t as

$$u_k(t) = \log \frac{p(z_k = 1 | z_{\setminus k})}{p(z_k = 0 | z_{\setminus k})} \ , \tag{2.14}$$

where $z_k = z_k(t)$ and $z_{\setminus k}$ are the values $z_i(t)$ of all other units with $i \neq k$. For the Boltzmann distribution (Eq. 2.6), the NCC requires neuron k to have the membrane potential

$$u_k(t) = b_k + \sum_{i=1}^{K} W_{ki} z_i(t) \ , \tag{2.15}$$

where b_k is the bias of neuron k and regulates its excitability, W_{ki} is the synaptic strength between neuron k and i and $W_{ki} z_i(t)$ represents the time course of the postsynaptic potential in neuron k caused by a firing of neuron i. To specify exactly when the neuron has fired during the time interval $(t-\tau, t]$, additional non-binary variables $(\zeta_1, ..., \zeta_K)$ are introduced. It is because of these auxiliary variables that the sampling process of the AMNs has the Markov property: if a neuron only had z_k as a state variable, it would not 'know' when to exit a refractory state.

In discrete time and for a neuron model with an absolute refractory mechanism , the dynamics of ζ_k are defined in the following way: ζ_k is set to τ (which is the refractory period) when neuron v_k fires, and decays by 1 in each subsequent discrete time step (see figure 2.2).

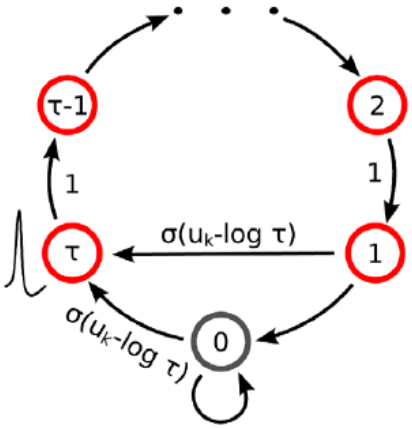


Figure 2.2.: A schematic of the internal state variable ζ_k of a spiking neuron v_k with an absolute refractory period. During the refractory period, ζ_k decays by 1 in each subsequent discrete time step and is reset to τ when the neuron fires. The neuron can only fire in the resting state $\zeta_k = 0$ and in the last refractory state $\zeta_k = 1$, with a probability defined by a logistic function. (Image is taken from *Buesing et al.*, 2011).

The neuron can only spike if $\zeta_k \leq 1$ and the spiking probability is defined by

$$p\left[z_k(t) = 1 | \zeta_k(t) \in \{0, 1\}, u_k(t)\right] = \sigma(u_k - \log \tau) \ , \tag{2.16}$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the logistic function. To better illustrate how z_k, ζ_k and u_k evolve during a NS process in discrete time, traces of the three variables of one neuron are shown in figure 2.3.

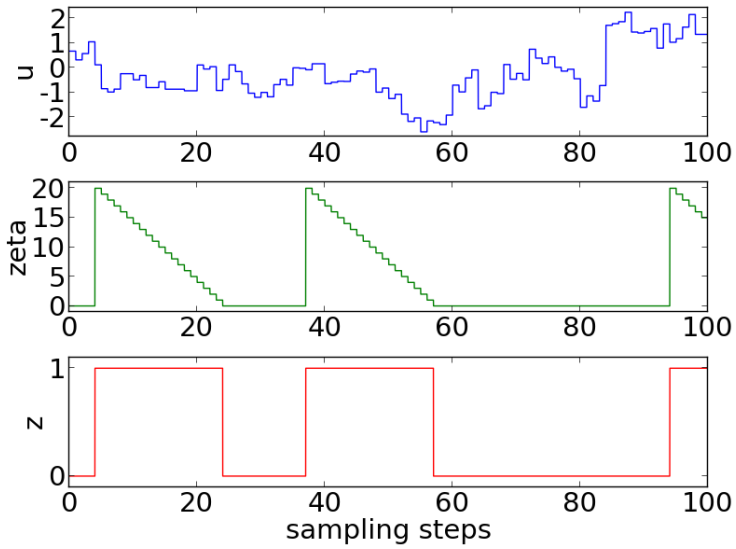


Figure 2.3.: The figure shows an example of the trace of z, ζ and u of a single selected AMN for 100 sample steps in an NS process in discrete time with randomly selected weights and biases. The refractory period τ is chosen to be 20 and total neuron number is 100.
While the membrane potential u is updated at every discrete time step, ζ decreases from τ to 1 in a fixed manner during the refractory period, and is reset to τ at $z = 1$ or $z = 0$ with a probability depending on the value of u at that time. The state of the neuron z is set to 1 during the refractory period, otherwise 0.

2.4. LIF sampling

The sampling dynamics of AMN networks depends on the stochastic nature of their units. Seemingly in contrast to that, in vitro experiments have shown the largely deterministic nature of single neurons. Besides, microscopic models of neural circuits consisting of biologically plausible neuron models typically rely on deterministic dynamics of their constituents: a neuron spikes when its membrane potential is above a certain threshold, rather than according to a certain probability. An often-used model with such properties is the leaky integrate-and-fire (LIF) neuron model, described by the ODE

$$C_{\mathrm{m}} \frac{du_k}{dt} = g_{\mathrm{l}}(E_{\mathrm{l}} - u_k) + I_k \ , \tag{2.17}$$

with capacitance C_{m}, leak potential E_{l}, leak conductance g_{l} and input current I_k. The spiking condition is deterministic: When u_k crosses a threshold ϑ from below, a spike is emitted and u_k is reset to ϱ for a refractory period τ_{ref}. For conductance-based synapses,

the synaptic input current injected into a neuron is typically modelled as

$$\frac{dI_k^{\mathrm{syn}}}{dt} = -\frac{I_k^{\mathrm{syn}}}{\tau_{\mathrm{syn}}} + \sum_{\mathrm{syn}\,i} \sum_{\mathrm{spk}\,s} w_{ki}\,(E_i^{\mathrm{rev}} - u_k)\delta(t - t_s)\;, \tag{2.18}$$

with the synaptic time constant τ_{syn}, the synaptic weight w_{ki} and the reversal potential of the ith synapse E_i^{rev}. It was demonstrated (*Petrovici et al.*, 2013) that in a spiking noisy environment, by describing the spike response as a first-passage time (FPT) problem, the dynamics of a single LIF neuron will exhibit the stochastic features required by neural sampling and therefore are able to implement sampling from well-defined probability distributions. This directly implies the possibility to build an LIF-neuron-based BM which can perform stochastic inference on the same class of probability distributions as the BM based on AMNs.

2.4.1. Deterministic neurons in a noisy spiking environment

In a noisy spiking environment, the total input current I_k to a neuron can be partitioned into recurrent synaptic input, diffuse synaptic noise and additional external currents: $I_k = I_k^{\mathrm{rec}} + I_k^{\mathrm{noise}} + I_k^{\mathrm{ext}}$. Throughout the analysis of individual neurons, I_k^{rec} and I_k^{ext} are set to zero.

When a conductance-based neuron receives enough synaptic stimulation, it enters a so-called high-conductance state (HCS), characterized by accelerated membrane dynamics. In a high input rate regime, the equation governing the membrane potential can then be written as

$$\tau_{\mathrm{eff}} \frac{du}{dt} = u_{\mathrm{eff}} - u \tag{2.19}$$

$$u_{\mathrm{eff}} = \frac{I^{\mathrm{ext}} + g_{\mathrm{l}} E_{\mathrm{l}}}{\langle\, g_{\mathrm{tot}} \,\rangle} + \frac{\sum_i g_i^{\mathrm{noise}} E_i^{\mathrm{rev}}}{\langle\, g_{\mathrm{tot}} \,\rangle}\;, \tag{2.20}$$

with $\langle\,\cdot\,\rangle$ denoting the mean and g_i^{noise} representing the total conductance at the ith synapse. The membrane time constant $\tau_m = C_{\mathrm{m}}/g_{\mathrm{l}}$ in Eq. 2.17 is replaced by an effective time constant $\tau_{\mathrm{eff}} = C_{\mathrm{m}}/g_{\mathrm{tot}}$, with the total conductance g_{tot} subsuming both leakage and synaptic conductance. In a first-order approximation, τ_{eff} can be considered very small in the HCS, resulting in $u \approx u_{\mathrm{eff}}$, with the effective potential u_{eff} simply being a linear transformation of the synaptic noise input.

Based on an approach by *Ricciardi and Sacerdote* (1979), *Petrovici et al.* (2013) have shown that, if stimulated by a large number of uncorrelated spike sources, the synaptic current I^{noise} - and therefore, also u_{eff} - can be described as an Ornstein-Uhlenbeck (OU) process

$$du(t) = \theta \cdot (\mu - u(t)) + \Sigma \cdot dW(t)\;, \tag{2.21}$$

with parameters

$$\theta = \frac{1}{\tau_{\text{syn}}} \tag{2.22}$$

$$\mu = \frac{I^{\text{ext}} + g_{\text{l}} E_{\text{l}} + \Sigma_i \nu_i w_i E_i^{\text{rev}} \tau_{\text{syn}}}{\langle\, g_{\text{tot}} \,\rangle + \Sigma_i \nu_i w_i \tau_{\text{syn}}} \tag{2.23}$$

$$\Sigma^2 = \sum_i \nu_i w_i^2 (E_i^{\text{rev}} - \mu)^2 \tau_{\text{syn}} / \langle\, g_{\text{tot}} \,\rangle \ , \tag{2.24}$$

where ν_i represents the input rate at the ith noise synapse.

2.4.2. The activation function as an FPT problem

In the AMN, the spiking probability (2.16) can be viewed as an activation function

$$p(z = 1) = \sigma(v) := [1 + \exp(-v)]^{-1} \ . \tag{2.25}$$

An activation function of the deterministic LIF neuron in a spiking noisy environment can also be derived (*Petrovici et al.*, 2013). Analogously to the AMN, we define the refractory state of a neuron as $z = 1$. For neuron membrane dynamics with a reset mechanism, two modes of firing can be observed: the "bursting" mode, where the effective membrane potential after the refractory period is still above threshold, and the freely evolving mode, where the neuron does not spike again immediately after the refractory period. Denoting the relative occurrence of burst lengths n by P_n and the average duration of the freely evolving mode that follows an n-spike-burst by T_n, we can identify the following relation:

$$p(z = 1) = \frac{\sum_n P_n \cdot n \cdot \tau_{\text{on}}}{\sum_n P_n \cdot (n \cdot \tau_{\text{on}} + T_n)} \ . \tag{2.26}$$

Given the parameters of the associated OU process, a recursive expression for P_n and T_n can be derived, which ultimately allows the calculation of $p(z = 1)$:

$$\begin{aligned} P_n &= p(u_n < \vartheta, ..., u_1 \geq \vartheta | u_0 = \vartheta) \\ &= \left(1 - \textstyle\sum_{i=1}^{n-1} P_i\right) \int_\vartheta^\infty du_{n-1} p(u_{n-1} | u_{n-1} \geq \vartheta) \\ &\quad \left[\int_{-\infty}^{\vartheta} du_n p(u_n | u_{n-1})\right] \end{aligned} \tag{2.27}$$

$$\begin{aligned} T_n &= \int_\vartheta^\infty du_{n-1} p(u_{n-1} | u_{n-1} \geq \vartheta) \\ &\quad \left[\int_{-\infty}^{\vartheta} du_n p(u_n | u_n < \vartheta, u_{n-1}) \,\langle\, T(\vartheta, u_n) \,\rangle\right] \ . \end{aligned} \tag{2.28}$$

The transfer function $p(u_n|u_{n-1})$ is the Green's function of the OU process for $t = \tau_{\text{ref}}$:

$$p(u_n | u_{n-1}) = C e^{-\frac{\theta}{\Sigma^2}\left[\frac{(u_n - (u_{n-1} - \mu)\exp(-\theta\tau_{\text{ref}}) - \mu)^2}{1 - \exp(-2\theta\tau_{\text{ref}})}\right]} \ , \tag{2.29}$$

with the normalization $C = \sqrt{\theta / \pi \Sigma^2 (1 - e^{-2\theta\tau_{\text{ref}}})}$. $T(u_b, u_a)$ denotes the time the membrane needs to reach u_b starting from u_a. To improve the prediction of the activation

function, one needs to take into account small, but finite τ_{eff}, in which case the membrane potential no longer directly follows the input current, but is low-pass-filtered value due to a non-zero τ_{eff}. By using an expansion in $\sqrt{\tau_{\text{eff}}/\tau_{\text{syn}}}$, a first-order correction to the FPT can be calculated (see *Brunel and Sergi* (1998)):

$$\langle\, T(\vartheta, u)\,\rangle = \tau\sqrt{\pi} \int_{\frac{u-\mu}{\sigma}}^{\frac{\vartheta_{\text{eff}}-\mu}{\sigma}} dx \exp(x^2)[\text{erf}(x) + 1]\,, \tag{2.30}$$

with the effective threshold $\vartheta_{\text{eff}} \approx \vartheta - \zeta(\frac{1}{2})\sqrt{\frac{\tau_{\text{eff}}}{2\tau_{\text{syn}}}}$, where ζ denotes the Riemann Zeta function. A comparison of the predicted $p(z = 1)$ with results from a numerical simulation is shown in Fig. 2.4.

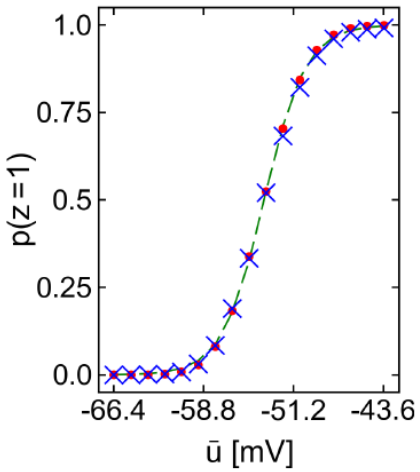


Figure 2.4.: The activation function of an LIF-neuron. Theoretical prediction (red) vs. simulation results (blue). A logistic function $\sigma(\bar{u})$ (green) is fitted to the prediction. (Image is taken from *Petrovici et al.*, 2013).

For the translation from the LIF neuron to the AMN,

$$v = (\bar{u} - \bar{u}^0)/\alpha\,, \tag{2.31}$$

where $\bar{u}^0$ denotes the value of $\bar{u}$ for which $p(z = 1) = \frac{1}{2}$ and α represents a scaling factor between the two domains. At this point, one can conclude that a single LIF neuron in a spiking noisy environment can closely reproduce the activation function (2.25).

2.4.3. Sampling via recurrent networks of LIF neurons

In addition to the discussed synaptic noise stimulus, an LIF neuron k in a recurrent network receives synaptic currents I_k^{rec} from other network neurons. Based on the

activation function derived in the previous section, an unconnected LIF neuron exhibits a well-defined spiking probability. For Boltzmann distributions as defined in (2.6), LIF neurons in a spiking noisy environment can closely approximate the logistic activation function defined by (2.25) if the mean membrane potential $\bar{u}_k$ is mapped in the LIF domain according to the linear translation in (2.31).

Using (2.31) to translate the bias b_k to a modification in the mean $\bar{u}_k$ (implementable by, e.g., a bias current or a modified leak potential) and estimating the impact of a pre-synaptic spike on the post-synaptic neuron through conductance-based synapses of weight w_{kj} , the following parameter translations between the abstract and the LIF domain can be written down (*Petrovici et al.*, 2013):

$$b_k = (\bar{u}_k^b - \bar{u}_k^0)/\alpha \tag{2.32}$$

$$W_{kj} = \frac{1}{\alpha C_\mathrm{m}} \frac{w_{kj}(E_{kj}^\mathrm{rev} - \mu)}{1 - \frac{\tau_\mathrm{syn}}{\tau_\mathrm{eff}}} \left[\tau_\mathrm{syn}(e^{-1} - 1) - \tau_\mathrm{eff} \left(e^{-\frac{\tau_\mathrm{syn}}{\tau_\mathrm{eff}}} - 1 \right) \right] . \tag{2.33}$$

where $\bar{u}_k^b$ denotes the mean free potential $\bar{u}_k$ in Fig. 2.4, and E_{kj}^rev is the reversal potential for synapse W_{kj}. The idea of Eq. 2.33 is to match the integrals of individual postsynaptic potentials on v_k and $\bar{u}_k$. Furthermore, short-term synaptic depression was employed to approximate the theoretically optimal rectangular PSP shape for consecutive spikes (bursts).

The above formalism allows the precise individual configuration of every neuron and afferent synapse in the network, as long as the activation function of individual neurons can be measured. This renders parameter variations almost completely irrelevant, requiring precision only for leak potentials and synaptic weights. Evidently, this kind of robustness offers major advantages for an implementation in analog neuromorphic hardware, which always exhibits some degree of fixed-pattern noise. A software framework that implements this automated tuning protocol (*Petrovici*, 2014, in preparation) has been used throughout this work.

In order to translate synaptic interactions in the form of (W_{kj}, w_{kj}) and a neuronal excitability $(b_k, \bar{u}_k^b)$ between the two domains (AMN, LIF neuron), the parameters $\bar{u}_0^b$ and α can be read out from the form of the activation function. Its inflection point $\bar{u}_0^b$ and slope α can then be used according to Eq. 2.32, 2.33.

Since the performance of neural sampling has already been evaluated in (*Buesing et al.*, 2011) for implementations with AMNs for several inference tasks, this model will be employed as a benchmark for LIF-based implementations. For the presented inference tasks in the following chapters, this will be the general working protocol.

3. Perceptual multistability as a probabilistic inference task

Based on the theoretical foundations from the previous chapter, in the following sections we will evaluate the sampling performance of leaky integrate- and-fire (LIF) neurons applied to a "fully visible" BM (without hidden units). As a specific application, we choose a perceptual multistability setup, which has already been discussed in (*Buesing et al.*, 2011) for an implementation with abstract model neurons (AMNs). These results will be used as benchmarks for evaluating the LIF results. At first, we perform the training of an AMN-based BM to model perceptual multistability, then translate the trained model parameters to the LIF-based BM to compare the results of both implementations. Due to the characteristics of the conductance-based LIF model, a straightforward weight translation (as proposed in section 3.2.1) will introduce additional deviations to the LIF network, impairing its sampling performance. To solve this problem, a method for rescaling synaptic weights is proposed, which improves the sampling performance visibly, but still not quite to the level of the AMN-based implementation. Therefore, the next step is to train the LIF-based BM directly. To achieve this, we use a CD learning algorithm based on LIF samplers, and the result after the training of the model proves its feasibility. Additionally, the good results achieved with this training method when applied to LIF neurons with noised parameters shows its potential for a future implementation on neuromorphic hardware.

3.1. Modeling perceptual multistability with AMN-based BMs

Perceptual multistability is a phenomenon evoked by ambiguous sensory stimuli, such as a 2D picture (e.g.,a Necker cube, see Fig. 3.1) that will cause different consistent 3D interpretations. When under such stimulus, the perception of humans and non-human primates will switch between different self-consistent global percepts, rather than produce a superposition of different possible percepts (*Buesing et al.*, 2011).

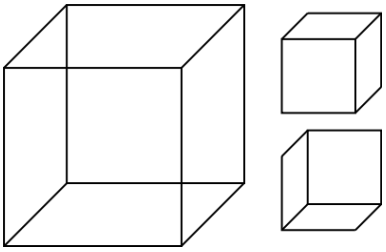


Figure 3.1.: Ambiguous sensory stimulus: the Necker cube (on the left). A perceptual switch of the perceived orientation can be triggered voluntarily, but a superposition of both percepts can never happen.

A standard experimental paradigm for studying this effect is binocular rivalry, where different images are presented to the left and right eye. A typical pair of stimuli are the two images shown in Fig. 3.2.

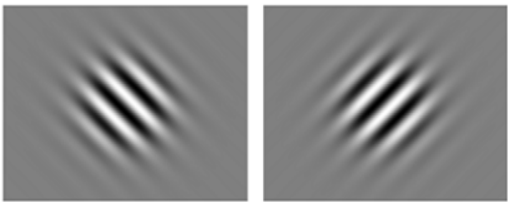


Figure 3.2.: Typical visual stimuli for the left and right eye in binocular rivalry experiments. When presented with such contradictory stimuli, human perception switches between the two presented orientations. (Image is taken from *Buesing et al.*, 2011)

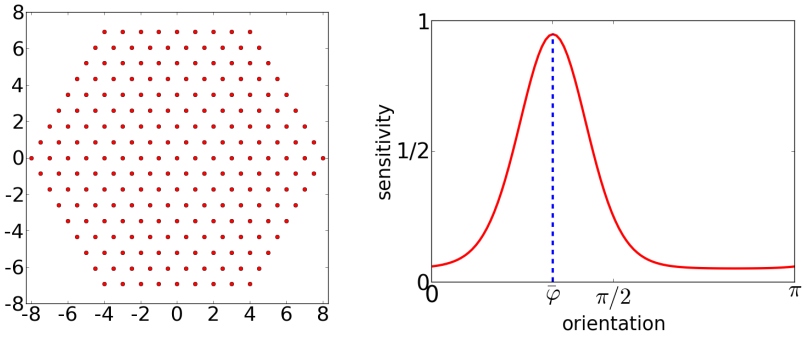
Simulations by (*Buesing et al.*, 2011) suggested that perception can be interpreted as probabilistic inference carried out by MCMC sampling. In particular, they have shown how these phenomena can be reproduced within the neural sampling framework.

3.1.1. Experimental setup

To model perceptual multistability, 217 abstract model neurons were arranged on a hexagonal grid (see Fig. 3.3a). Neighboring units had distance 1 and any two neurons with distance ≤ 8 were connected. Each neuron was randomly assigned a preferred orientation φ_k and all the preferred orientations were chosen to cover the entire interval $[0, \pi)$ with equal spacing. A corresponding tuning curve centered around the neuron's preferred orientation (see Fig. 3.3b) defined its firing probability under external stimuli

$$V_k(\varphi) = \nu_0 + C \exp[\kappa \cos(2(\varphi - \bar{\varphi}_k)) - \kappa] \,, \tag{3.1}$$

where C denotes the sensitivity contrast, κ denotes the peakedness and ν_0 denotes the base sensitivity. In this particular implementation, the values of C, κ and ν_0 were chosen to be 0.9, 3 and 0.05, respectively.



(a) (b)

Figure 3.3.: Experimental setup.
(a): 217 AMNs arranged on a hexagonal grid. The distance between neighboring units is 1 and any two neurons with distance ≤ 8 are connected. Each neuron is randomly assigned a preferred orientation and its firing probability under external stimuli (a global input orientation) is defined by a tuning curve (b).

3.1.2. Training

Prior to training, the initial weights and biases were set to 0. The employed learning algorithm was CD and the parameters were updated as follows:

$$\begin{aligned} \Delta W_{ki} &= \eta_{ki}(\tilde{z}_k\tilde{z}_i - z_k^* z_i^*) \\ \Delta b_k &= \eta(\tilde{z}_k - z_k^*) \ , \end{aligned} \tag{3.2}$$

where $\tilde{\mathbf{z}}$ denotes posterior samples (network states under the influence of present input), $\mathbf{z}^*$ denotes approximate prior samples (network states in the absence of stimuli), η denotes the learning rate and η_{ki} equals η if ν_k and ν_i were connected, otherwise 0. The learning rate was chosen to be constant at 10^{-4} during training.

The posterior sample was generated in the following way:

1. A global input orientation φ was uniformly drawn from $[0, \pi)$.
2. Each neuron then fired independently with probability $p(z_k = 1) = V_k(\varphi)$.
3. The resulting network state was taken as a posterior sample.

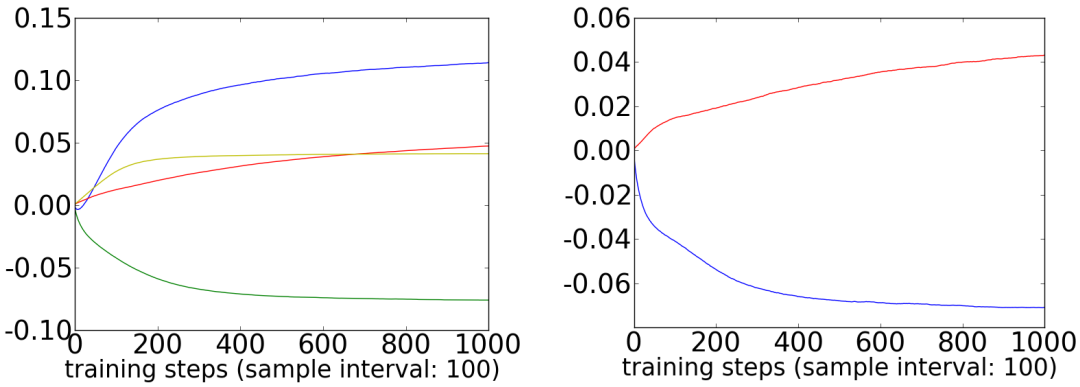
The approximate prior sample was obtained by:

1. Randomly choosing ζ_k uniformly in $\{1, \&, \tau\}$ if $\tilde{z}_k = 1$ and $\zeta_k = 0$ otherwise.
2. After evolving freely for τ steps from the posterior state, the network state was taken as an approximate prior sample.

In the experiment, τ was chosen to be 20. The weights and biases were updated immediately after obtaining the two samples. In total, we trained the network for 10^5 steps.

3.1.3. Results

As expected, after learning, the network connections showed high specificity: neurons with similar preferred orientations were connected with excitatory weights ($W_{ki} = 0.117 \pm 0.049$), and dissimilar orientations with inhibitory weights ($W_{ki} = -0.076 \pm 0.042$) (see Fig. 3.4a). The preferred orientations $\bar{\varphi}_i$ and $\bar{\varphi}_j$ were defined as similar if $V_i(\bar{\varphi}_j) - \nu_0 = V_j(\bar{\varphi}_i) - \nu_0 > 0.5C$, otherwise dissimilar. The mean neuron biases converged to -0.071 ± 0.044 (see Fig. 3.4b).



(a) (b)

Figure 3.4.: Evolution of mean weights and biases during 10^5 training steps with a constant learning rate $\eta = 10^{-4}$.
(a): Mean of weights of similar preferred orientation (SPO, blue), standard deviation of weights of SPO (red), mean of weights of dissimilar preferred orientation (DPO, green), standard deviation of weights of DPO (yellow). After training, neurons with similar (dissimilar) preferred orientations are connected with mean positive (negative) weights.
(b): Mean of biases (blue), standard deviation of biases (red). After training, neuron mean bias converged to a negative value.

To understand the high-dimensional network states more intuitively, their distribution was evaluated by a population vector plot and a plot of the marginal distribution of neuron firing probabilities. The population vector is a 2-dimensional projection of the high-dimensional network state

$$\mathbf{x} = (x_0, x_{\pi/4}) = \sum_{k=1}^{K} z_k \cdot (\cos 2\bar{\varphi}_k, \sin 2\bar{\varphi}_k) \; . \tag{3.3}$$

According to the definition, the orientation of the population vector will correspond to the dominant orientation of the percept, and its distance from the origin represents the percept strength. A network state with a large strength of percept is defined as a coherent state.

The marginal firing distribution was obtained by running a simulation for a long time. For our task, the network was run for $5 \cdot 10^5$ sampling steps. We can see that when the network evolved freely, the population vectors (see Fig. 3.5a) show an arbitrary orientation, as expected. Moreover, they are concentrated around large radii, which denotes network states with strong coherence. The marginal distribution plot (see Fig. 3.5b) of neuron firing probabilities is basically an even distribution with fluctuations of about 10%, indicating that all neurons share more or less the same firing probability.

To model binocular rivalry, the network was modified by clamping 8 neurons to emulate the grating stimuli in Fig. 3.2. Two neurons with $\bar{\varphi}_k \approx \pi/4$ and two with $\bar{\varphi}_k \approx 3\pi/4$ were clamped to $z = 1$. Additionally, two neurons with $\bar{\varphi}_k \approx 0$ and two with $\bar{\varphi}_k \approx \pi/2$ were clamped to $z = 0$. More specifically, neurons were clamped by setting their biases to extreme values (eg. 50 for clamped to 1 and -50 for clamped to 0).

The result (see Fig. 3.5c) shows that the network spends most of the time in states that correspond to one of the two input orientations which were facilitated via clamping. The black line shows the trace of network state during a perceptual switch. The marginal firing probability plot (see Fig. 3.5d) shows a symmetric distribution pattern, where neurons with preferred orientation closer to one of the clamped orientations fire with higher probability.

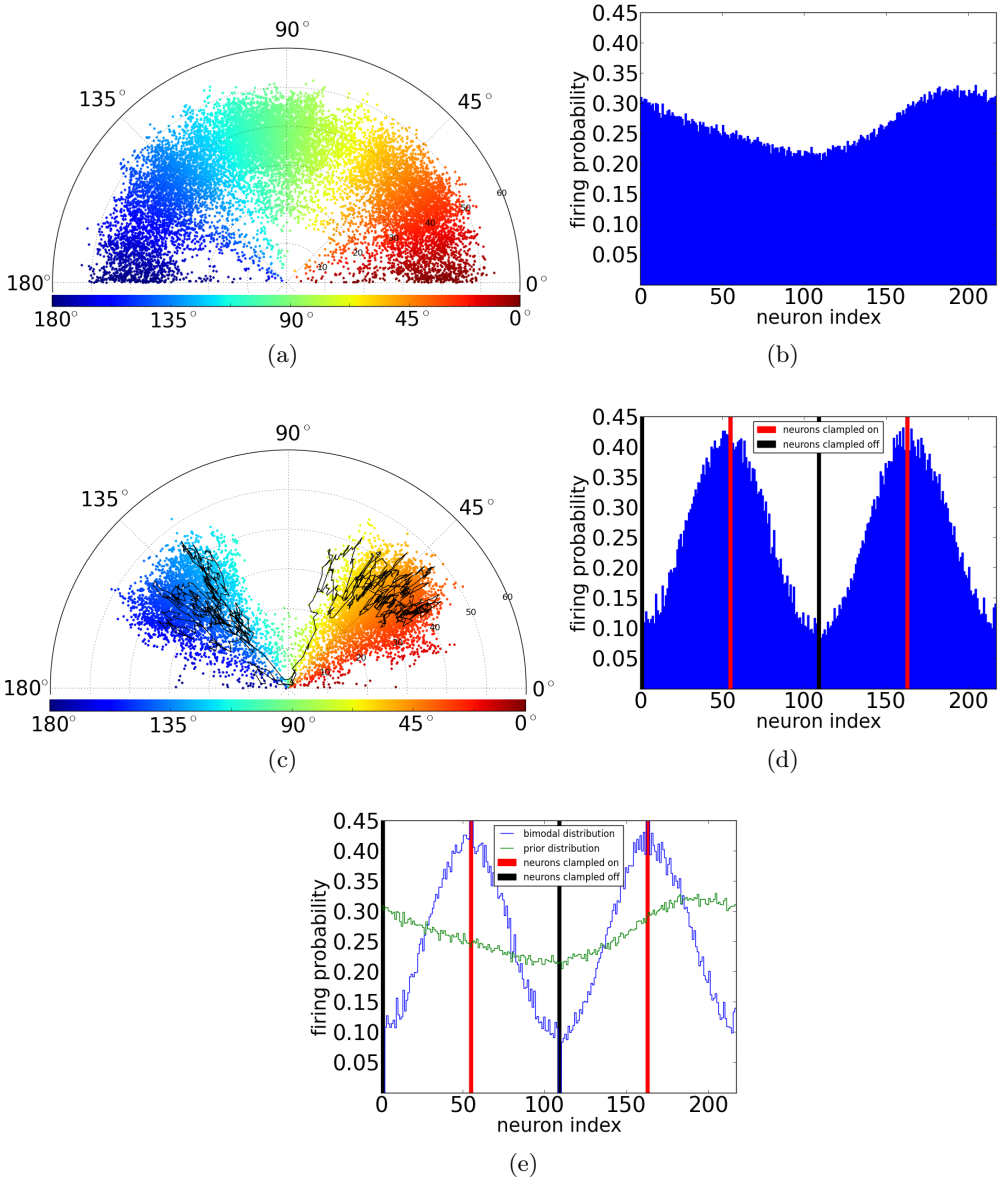
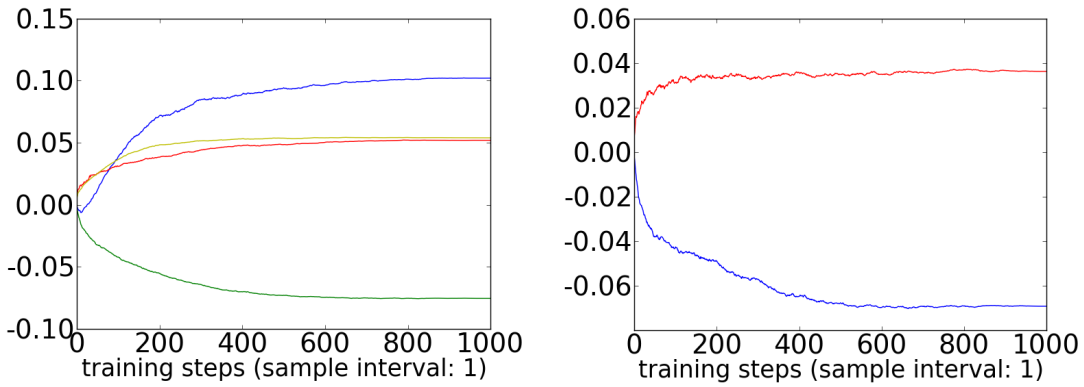


Figure 3.5.: Results after 10^5 training steps with a constant learning rate $\eta = 10^{-4}$. (a): Population vectors (PV) plot of $2 \cdot 10^4$ consecutive sampling steps ($N_{steps} = 2 \cdot 10^4$). Training leads to strongly coherent states. (b): Marginal firing distribution (MFD) of prior states. Neurons are arranged along the x-axis according to the value of their preferred orientation: preferred orientation goes up in positive direction of the axis. $N_{steps} = 5 \cdot 10^5$. (c): PV plot of clamped states. $N_{steps} = 2 \cdot 10^4$. The black line shows the evolution of the network state for 1000 N_{steps} in the proximity of a perceptual switch. (d): MFD of clamped states. Neurons are arranged in the same way as in (b). $N_{steps} = 5 \cdot 10^5$. (e): A comparison between two MFDs shows that the clamped input changes the firing activity of the network. Firing frequencies of neurons with SPO as the neurons clamped to 1 are increased. Conversely, firing frequencies of neurons with SPO as the neurons clamped to 0 are decreased.

The training took about 1 hour (on IGNATZ[1]) with a constant learning rate during training. We were able to reduce this training time by a factor of 100 by the learning algorithm with an exponentially decreasing learning rate:

$$\eta(n) = (10^{-4} - 10^{-2} + 1)^{\frac{n}{1000}} + 10^{-2} - 1 \ , n = 0, 1, 2...N$$

With this setup, η decreased from 10^{-2} to 10^{-4} in $N = 10^3$ time steps. The mean weights and biases still converged to almost the same values (see Fig. 3.6), i.e., neurons with similar (and also dissimilar) preferred orientations shared generally the same weight strengths as when trained with constant learning rates.

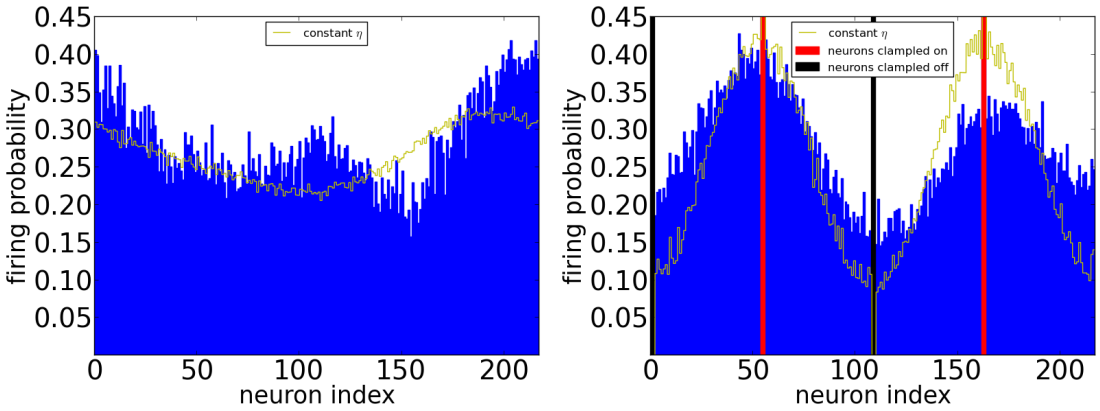


(a) (b)

Figure 3.6.: Evolution of mean weights and biases during 10^3 training steps with exponentially decreasing learning rates. After training, the mean weights and biases converged to almost the same values as when training with a constant learning rate (see Fig. 3.4).
(a): Mean of weights of similar preferred orientation (SPO, blue), standard deviation of weights of SPO (red), mean of weights of dissimilar preferred orientation (DPO, green), standard deviation of weights of DPO (yellow).
(b): Mean of biases (blue), standard deviation of biases (red).

After training, the marginal firing probability distribution of prior states (see Fig. 3.7a) exhibited more fluctuation than Fig. (3.5b). For bimodal states (see Fig. 3.7b), the distribution was less symmetric compared to the result obtained with a constant learning rate. The fluctuations can be explained by the more "crude" training algorithm. The decrease in symmetry is likely due to the average times the network needs to propagate from one mode to the others. Since such a propagation time is in the order of seconds, (1 sampling step[2] is defined to be equal to 1 ms) (*Buesing et al.*, 2011), longer simulation might yield a more symmetric distribution pattern.

[1]IGNATZ: AMD Phenom(tm) II X4 965 Processor
[2]update the state of all units for one time



(a) (b)

Figure 3.7.: Marginal firing probability distributions after 10^3 training steps with learning rates exponentially decreasing from 10^{-2} to 10^{-4}.
(a): MFD of prior states. The distribution exhibits more fluctuation than when training with a constant learning rate. $N_{steps} = 5 \cdot 10^5$. (b): MFD of clamped states. The distribution exhibits more fluctuation and is also not as symmetric as when training with a constant learning rate. $N_{steps} = 5 \cdot 10^5$.

The result can be improved by further training the model with a constant small learning rate to fine tune the model parameters. Also, more elaborate learning rate decreasing mechanisms might improve the uniformity of the network behavior.

We conclude that a decreasing learning rate can be an optimal choice when training large networks, although the result might be not as good as when training with a constant small learning rate. After training, the network states qualitatively showed the desired distributions in both the freely evolving and the clamped input cases. Next, we transfer the trained model parameters to the LIF-based BM.

3.2. Modeling perceptual multistability with LIF-based BMs

For the LIF-based BMs, we chose the set of neuron parameters (conductance-based) from table 3.1 (neuron parameters as defined in PyNN).

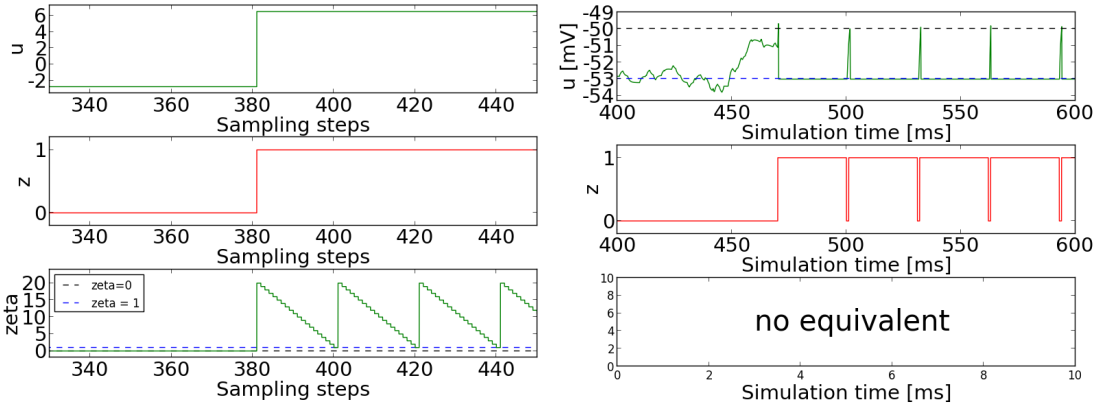
Name (PyNN)	Value	Units	Description
`v_rest`	-50.0	mV	Resting membrane potential u_{rest}
`cm`	0.2	nF	Capacity of the membrane C_{m}
`tau_m`	1.0	ms	Membrane time constant τ_{m}
`tau_refrac`	30.0	ms	Duration of refractory period τ_{ref}
`tau_syn_E`	30.0	ms	Decay time of the excitatory synaptic conductance $\tau_{\text{exc}}^{\text{syn}}$
`tau_syn_I`	30.0	ms	Decay time of the inhibitory synaptic conductance $\tau_{\text{inh}}^{\text{syn}}$
`e_rev_E`	0.0	mV	Reversal potential for excitatory input $E_{\text{exc}}^{\text{rev}}$
`e_rev_I`	-100.0	mV	Reversal potential for inhibitory input $E_{\text{inh}}^{\text{rev}}$
`v_thresh`	-50.0	mV	Spike threshold ϑ
`v_reset`	-50.001	mV	Reset potential after a spike u_{reset}
`i_offset`	0.0	nA	Offset current I_{off}

Table 3.1.: Neuron parameters for our software (NEURON) simulations.

The distance between reset potential and threshold is set to be small, which allows the membrane potential to jump to the threshold quickly after the refractory period. This enables a better correspondence between the firing statistics of the LIF samplers and the AMNs.

For example, during a period of continuous spiking, the AMN can enter the next refractory period immediately after the end of the last refractory period and the state of the neuron can always be 1 (as shown in Fig. 3.8a). However, the same situation, the membrane potential of the LIF neuron needs to first reach the threshold starting from the reset potential at the end of the last refractory period (see Fig. 3.8b). For this finite interval of time, the LIF neuron is in the wrong ($z = 0$) state. When the distance between the reset potential and the threshold is small enough, the jumping will take almost no time and the state of the LIF-neuron can almost always be 1, thereby yielding a better approximation of the AMN z-dynamics.

High-frequency Poisson noise was injected into the populations to make the neurons enter a high-conductance state. Each neuron had two independent Poisson sources, one excitatory and the other inhibitory. The rate of the noise was set to 0.4 kHz and the synaptic weights of the noise input were all set to 0.002 uS.



(a) AMN (b) LIF neuron

Figure 3.8.: Comparison of neuron states during continuous spiking.
(a): Traces of the membrane potential u, state z and auxiliary variable ζ of an AMN. During continuous spiking, the AMN is able to jump to the next refractory period immediately at the end the last refractory period while maintaining its state at 1. (b): Traces of membrane potential u and the state z of an LIF neuron with reset potential -53 mV and threshold -50 mV. During continuous spiking, the membrane potential needs to reach the threshold at the end of the last refractory period, the neuron state during the "jumping time" is 0. The time in the 0 state can be reduced if the distance between the threshold and the reset potential is set to be small.

3.2.1. Direct parameter translation to LIF neurons

We first applied ideal neuron parameters to all neurons, which means there is no added noise and all the neurons share identical parameters as in table 3.1. After the translation (see Eq. 2.32, 2.33) of the trained AMN parameters (we chose the model parameters trained for 10^5 times with a constant learning rate $\eta = 10^{-4}$) to the LIF domain, we first ran a free simulation of the network (with the NEURON simulator) to evaluate the prior distribution. The result can be seen in Fig. (3.9a) and (3.9b).

To emulate the grating stimuli for the binocular rivalry experiment, two LIF neurons representing $\bar{\varphi}_k \approx \pi/4$ ($\bar{\varphi}_k \approx 0$) and $\bar{\varphi}_k \approx 3\pi/4$ ($\bar{\varphi}_k \approx \pi/2$) were effectively clamped by a large increase (decrease) of their v_rest values to enforce permanent spiking (silence). The simulation result can be seen in Fig. (3.9c) and (3.9d).

A long simulation time is needed in order to obtain a stable marginal firing probability distribution of the network. Especially for the distribution of the bimodal states, since the model switches between the two modes stochastically, a longer simulation time is more likely to guarantee that the states of the model will show a more symmetric distribution. In the simulation of the LIF network, 1 ms corresponds to 1 sampling step

in the AMN experiment. The refractory time constant is 30 ms for the LIF neuron, for the AMN it is 20 sampling steps. Therefore, we chose a simulation time of $7.5 \cdot 10^5$ ms to obtain the marginal firing probability distribution of the model, which corresponds to the $5 \cdot 10^5$ sampling steps in the AMN experiment.

Originally we expected that after parameter translation, the LIF-based BM would exhibit almost the same distribution as the AMN-based BM. However, Fig. (3.9b) shows that the prior distribution is slightly different from the one obtained with AMN. The difference between the bimodal distributions (see Fig. 3.9d) is more significant, the relative peak height (maximum height minus minimum height) being approximately a factor of 2 below the one from the AMN based implementation.

Slight differences between states distributions are reasonable since the LIF-based BM and AMN-based BM are essentially two different models. However, previous work (*Petrovici et al.*, 2013) has shown that, in a small network (eg. 5 neurons), an LIF-based BM with direct parameter translation from the AMN-based BM can actually represent the target probability distribution with very high accuracy. From Fig. (3.9e) one observes that the two distinctive modes are significantly less pronounced in the LIF marginal distribution, with the clamping mechanism showing less impact on neighbouring neurons than the AMN-based network (see Fig. 3.5e). One possible explanation for this mismatch will be discussed in the following section.

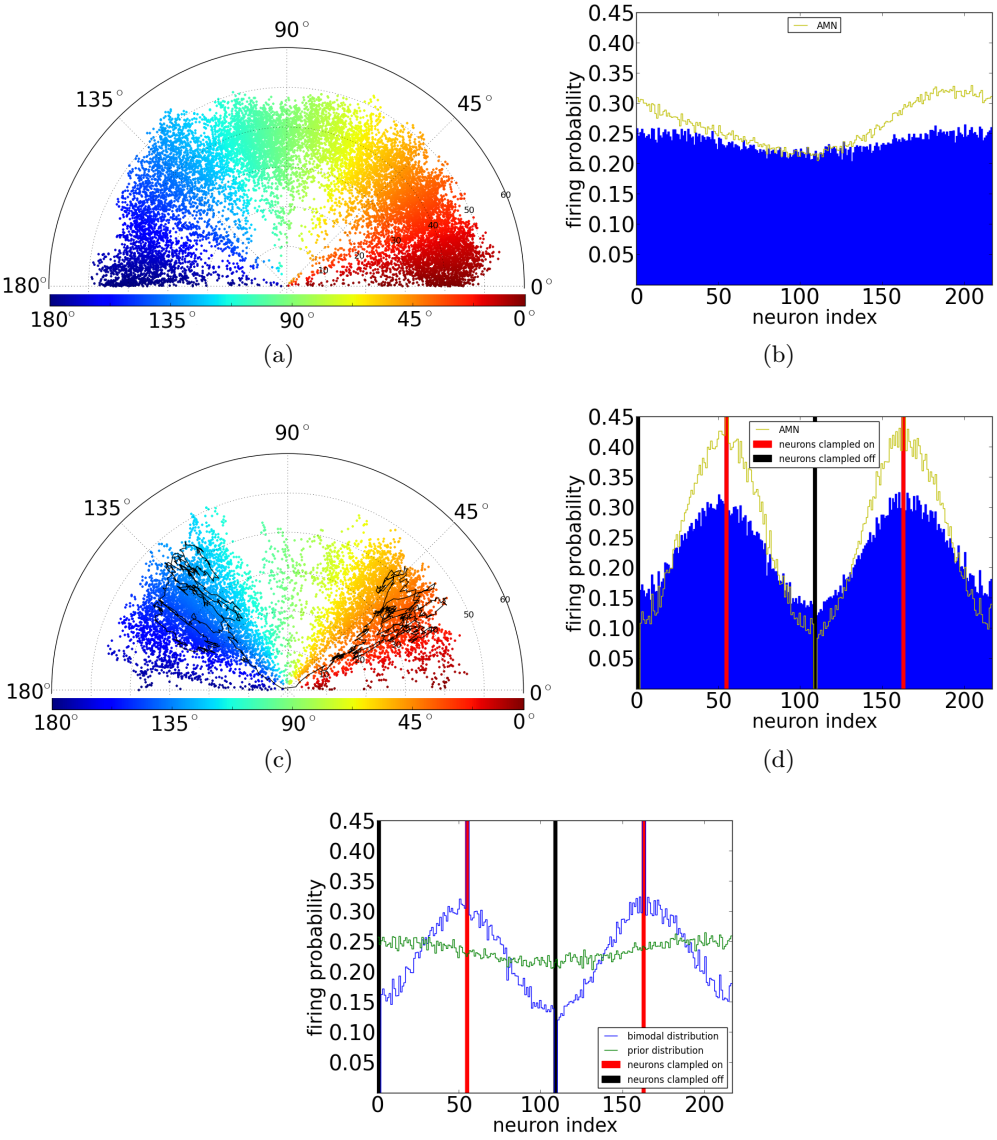


Figure 3.9.: Results for the LIF-based BM after direct parameter translation from trained model parameters of the AMN-based BM.
(a): PV plot of prior states. The model exhibits coherent states. Simulation time: $2 \cdot 10^4$ ms ($T_{sim} = 2 \cdot 10^4$ ms). (b): MFD of prior states. The firing probabilities distribution is below the corresponding AMN curve. $T_{sim} = 7.5 \cdot 10^5$ ms. (c): PV plot of clamped states. $T_{sim} = 2 \cdot 10^4$ ms, the black line shows the evolution of the network state for 1000 ms around a perceptual switch. (d): MFD of clamped states. The valleys and hills are not as steep as the corresponding AMN curve. $T_{sim} = 7.5 \cdot 10^5$ ms. (e): Comparison of the MFDs before and after clamping. The effect of clamping is significantly weaker than the in the AMN experiment (see Fig. 3.5e)

3.2.2. Problems in weight translation

To compare LIF-based networks with AMN-based networks, the LIF neurons are initialized according to the weight and bias translation rules in Eq. 2.32, 2.33. The weight translation can also be written as:

$$W_{kj} = \frac{1}{\alpha}\frac{w_{kj}\left(E_{kj}^{\mathrm{rev}}-\mu\right)}{C_{\mathrm{m}}-\tau_{\mathrm{syn}}g_{\mathrm{tot}}}\left[\tau_{\mathrm{syn}}(e^{-1}-1)-\tau_{\mathrm{eff}}\left(e^{-\frac{\tau_{\mathrm{syn}}}{\tau_{\mathrm{eff}}}}-1\right)\right] , \tag{3.4}$$

where $g_{\mathrm{tot}} = C_{\mathrm{m}}/\tau_{\mathrm{eff}}$. Theoretically, g_{tot} should be a sum of the leakage conductance g_{l} and all other synaptic conductances, including the conductance caused by excitatory and inhibitory Poisson input noises g_{e} and g_{i}, but also the conductance coming from inter-neuron synapses g^{n}. g_{e} and g_{i} can be calculated before the simulation. However, it is intractable to compute g_{n} before translating the BM parameters and actually running the simulation, since it would require the calculation of the joint distribution $p(\mathbf{z})$ for all possible combinations of states $\mathbf{z}$. Adding to the complexity, g^{n} is most likely not constant, but might vary significantly as a function of time.

In practice, g_{tot} is calculated by

$$\begin{aligned} g_{\mathrm{tot}} &= g_{\mathrm{e}} + g_{\mathrm{i}} + g_{\mathrm{l}} \\ g_{\mathrm{l}} &= C/\tau_{\mathrm{m}} \\ g_{\mathrm{e}} &= \sum_{\mathrm{exc\,syn}\,k} \nu_k w_k \tau_{\mathrm{exc}}^{\mathrm{syn}} \\ g_{\mathrm{i}} &= \sum_{\mathrm{inh\,syn}\,k} \nu_k w_k \tau_{\mathrm{inh}}^{\mathrm{syn}} , \end{aligned} \tag{3.5}$$

where τ_{m} is the membrane time constant, ν_k and w_k are the rate and weight of the corresponding Poisson input noise, and $\tau_{\mathrm{exc}}^{\mathrm{syn}}$, $\tau_{\mathrm{inh}}^{\mathrm{syn}}$ are the time constants of the excitatory and inhibitory noise input synapses.

In neural networks where inter-neuron spikes do not occur as frequently as the input noise (eg. network with few neurons, small weights and biases), the contribution of g^{n} compared to g_{tot} is small. In our setup, the network consists of 217 neurons and the number of afferent synapses from the network for each neuron ranges from 80 to 216. On this scale, the influence of g^{n} can no longer be ignored.

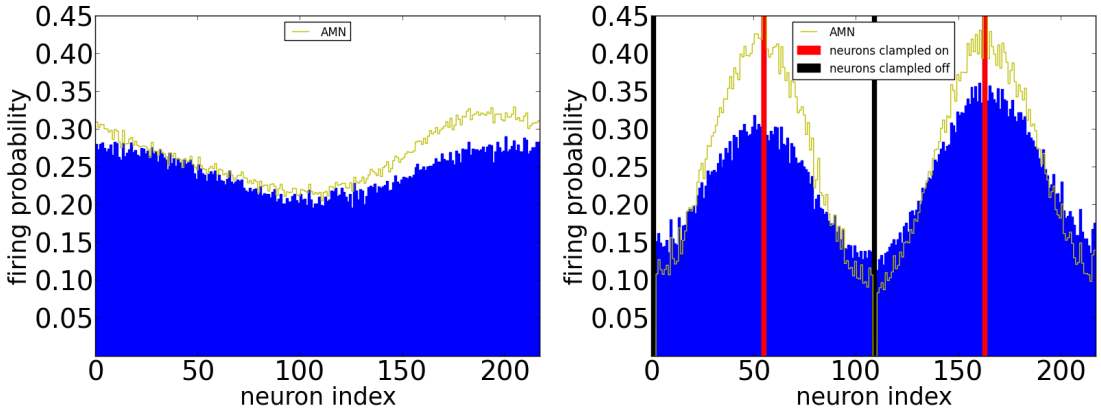
In order to calculate the average conductance interactions within the network for each neuron, we assume that all neurons fire with approximately constant rates. Under this assumption, we can calculate g^{n} by starting a simulation S and recording the spiking frequency for each neuron.

$$g_j^{\mathrm{n}} = \sum_{\mathrm{syn}\,i} \nu_i w_{ij} \tau_j^{\mathrm{syn}} , \tag{3.6}$$

where g_j^{n} denotes the sum of inter-neuron synaptic conductance of neuron j, w_{ij} is the strength of the synapse from neuron i to neuron j, ν_i is the spiking frequency of neuron i, and τ_j^{syn} is the synaptic time constant of neuron j. ν_i is calculated as the quotient of the total number of spikes divided by the total simulation time. A weight-rescale factor wrf_j^S for neuron j in simulation S can then be obtained:

$$\mathrm{wrf}_j^S = 1 + \frac{g_j^{\mathrm{n}}}{g_j^{\mathrm{tot}}} \, . \tag{3.7}$$

In the second step, we run a simulation S' with the same simulation time as S and for each neuron k, we multiply wrf_k^S to $g_{\mathrm{tot}k}$ during the weight translation. The result after applying this iterative weight-rescale method is in Fig. 3.10. We then take the model state from simulation S' as the result. A comparison of results from three models is shown in Fig. 3.11.



(a) (b)

Figure 3.10.: Results for the LIF-based BM with the weight rescaling method in the parameter translation. (a): The prior marginal distribution is closer to the one in the AMN-based network. $\mathrm{T_{sim}} = 7.5 \cdot 10^5$ ms. (b): MFD of the clamped states. The relative peak height also improves, although it does not quite reach the one measured in the AMN-based implementation. $\mathrm{T_{sim}} = 7.5 \cdot 10^5$ ms.

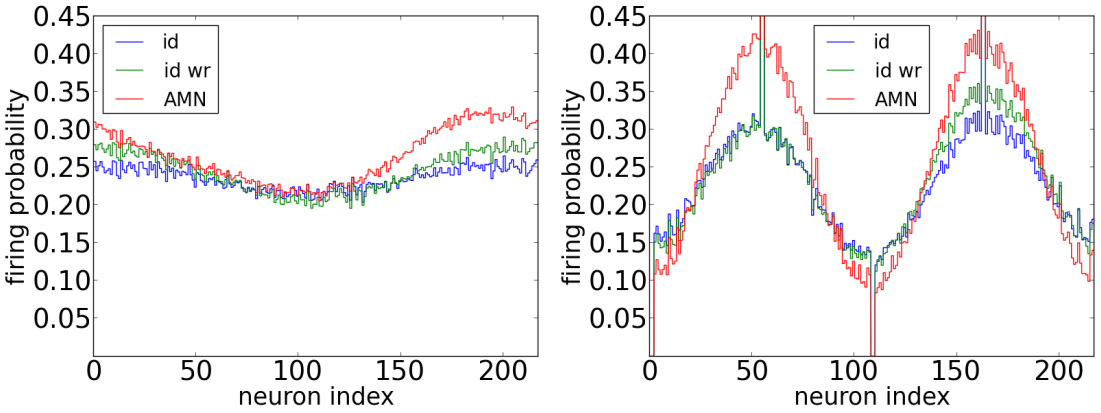


Figure 3.11.: Comparison of firing distributions: LIF-based BM with ideal neuron parameters with direct model parameter transfer (id), LIF-based BM with ideal neuron parameters with weight-rescale method (id wr), and the theoretical benchmark from the AMN-based network (AMN). Qualitatively, the weight-rescale procedure has slightly improved the LIF-based implementation.

The mean weight-rescale factor for a simulation time of $7.5 \cdot 10^5$ ms was calculated to be about 1.09, which means $g_n/g_{\text{tot}} \approx 0.09$. This yields a slightly better approximation of the neuron's firing probabilities to the AMN-benchmark. The prior marginal distribution is closer to the displayed AMN curve. The right mode of the bimodal distribution is lifted up and more similar to the AMN curve, while the left mode remains almost the same as before. This result could potentially be attributed to the average times the network needs to propagate from one mode to the others. Since such a propagation time is in the order of seconds (*Buesing et al.*, 2011), even longer simulation times than $\text{T}_{\text{sim}} = 7.5 \cdot 10^5$ ms need to be used to ensure converged marginal probabilities.

The result might be further improved if the weight rescaling method can be iterated more times, but it is also possible that it will not converge to the target distribution. Besides, adding iterating times is time-consuming, since the network needs to be reinitialized for every iteration. Another addition that might improve the result is to multiply the weight rescale factor to the biases as well. However, we did not spend much effort in the optimization of the weight rescaling, since its main purpose was to analyze the causes of the deviation of the LIF result from the theoretical benchmark, and was never meant to be an ultimate solution.

We conclude that the weight rescaling method does indeed improve the quality of the sampled distribution (i.e., the uniformity of the prior and the marked bimodality of the posterior). However, since it is probably not precise enough for more complex

tasks, we propose to move away from the translation paradigm and train the LIF-based networks directly.

3.2.3. LIF-based BMs with noised neuron parameters

In the neuromorphic hardware, fixed-pattern noise exists in neuronal circuitry. This results in neuron-to-neuron variability of the neuron parameters. In order to test the sampling performance of LIF units for such a setup, we added random noise to the neuron parameters (see table. 3.1) for each neuron.

Neuron parameter (PyNN syntax)	Noise value	Units
`tau_m`	0.1	ms
`e_rev_E`, `e_rev_I`, `v_rest`	2.0	mV
`v_reset`, `v_thresh`	0.5	mV
`tau_syn_E`, `tau_syn_I`	2.0	ms
`tau_refrac`	1.0	ms

Table 3.2.: Noise values for neuron parameters

The parameter noise was assumed as additive and uniformly distributed (around 0, with widths as given in table 3.2). The noise values were chosen according to personal communication with Mihai A. Petrovici and Marc-Olivier Schwartz. `tau_m` was chosen to be 2 ms and an error of 5% in the hardware was tolerated. The noise values of `e_rev_E`, `e_rev_I` and `v_rest` can be found in (*Schwartz*, 2013). For `v_reset` and `v_thresh`, repeated experiments on the hardware showed that it was possible to find about a quarter of all neurons with such parameter deviations. The noise values of `tau_syn_E` and `tau_syn_I` were estimated based on several early trials performed in 2013 by Petrovici and Schwartz. For `tau_refrac`, hardware experiments showed that about 80 neurons had this deviation when τ_{ref} was set to 30 ms.

To avoid that, after noising, the value of `v_thresh` becomes lower than `v_reset`, we changed the target value of `v_reset` to -53.0 mV. We ran two simulations with one applying the weight-rescale method and the other without, results can be seen in Fig. 3.12.

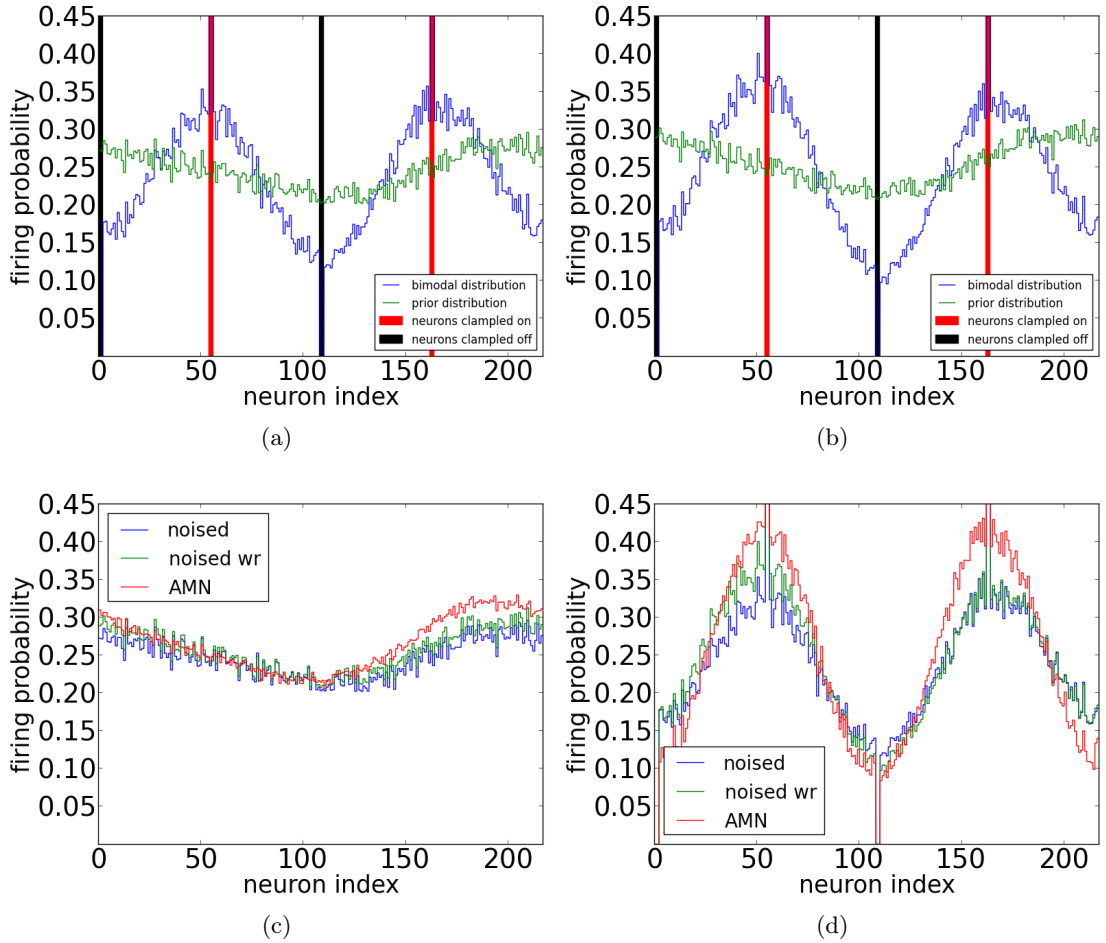


Figure 3.12.: Results for the LIF-based BM with noised neuron parameters.
(a): MFDs of direct parameter translation to the LIF-based BM with noised neuron parameters. $T_{sim} = 7.5{\cdot}10^5$ ms. (b): MFDs of parameter translation with weight-rescale method. $T_{sim} = 7.5 \cdot 10^5$ ms. (c): Comparison of MFDs. Both firing distributions for prior states are close to the AMN curve. (d): Comparison of MFDs. The weight-rescale method increases the height of the left peak of the firing distribution for clamped states, exhibiting a qualitative improvement towards the AMN curve.

Figure (3.12c) shows that both firing probability distributions for the prior states are very close to the AMN curve. Figure (3.12d) shows that the weight-rescale method increases the peak-to-valley ratio by 25%.

The performance of the self-calibrating implementation is quite evident, as the sampling performance is not affected by parameter variability (within the provided range). This is an indicator of the high accuracy of the model parameter translation mechanism for

each individual neuron, and shows the potential of implementing the LIF-based BM on neuromorphic hardware architectures, assuming they display reasonable parameter variability and offer sufficient configurability of synaptic weights and leak potentials.[3]

3.2.4. Direct training of LIF-based BMs

Although the weight rescaling method improved the LIF sampling performance with respect to the benchmark obtained via AMN-networks, training LIF-networks directly could implicitly resolve the problems introduced by the inherent differences between the AMN and LIF dynamics.

All the deviations are likely to be inherently amended when training the LIF-based BM directly, because the updating of model parameters now depends exactly on the model states itself. The CD algorithm guarantees that every time after updating the model parameters, the BM will move closer towards the target distribution. It is expected that in a long run the network will be improved in the right direction. However, small deviations still exist in the parameter translation between the LIF and the AMN domain, which might make the learning in the LIF-based BM not as efficient as in the AMN-based BM.

Before we start to train the LIF-based BM, additional setup needs to be considered and particular parameters need to be found. We will first discuss this setup and the method to find these particular parameters, and then introduce a detailed LIF-based CD learning algorithm based on these settings.

In the AMN network, the waiting time to get the prior state of the model was set to $\geq \tau$ in order to allow a decorrelation of the model states from its posterior states. Similarly, a decorrelation time also needs to be found for network of LIF neurons.

To explore the decorrelation of the network of LIF neurons as a function of time, we used the following method:

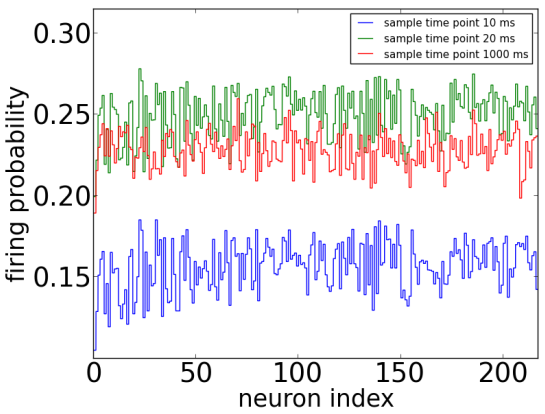
1. First, we ran multiple simulations of the same duration and recorded the states of all the neurons at time points uniformly distributed along the simulation time.
2. Then, states at the same point in time were summed over all the simulations and a firing probability distribution of all neurons was computed for each point in time.
3. Finally, the Euclidean distance (ED) between each firing distribution profile and a chosen benchmark was calculated.

Theoretically, a network of LIF neurons will decorrelate from its initial state after infinite simulation time, and the firing probability distribution after an infinitely large time interval would be the best benchmark. In practice, we chose the distribution at

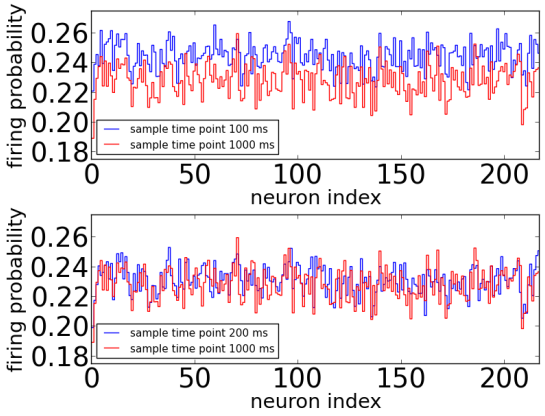
[3] the latter is actually not even needed, as the effective leak potential can be tuned with the synaptic weights of the noise stimulus.

1000 ms as the benchmark.

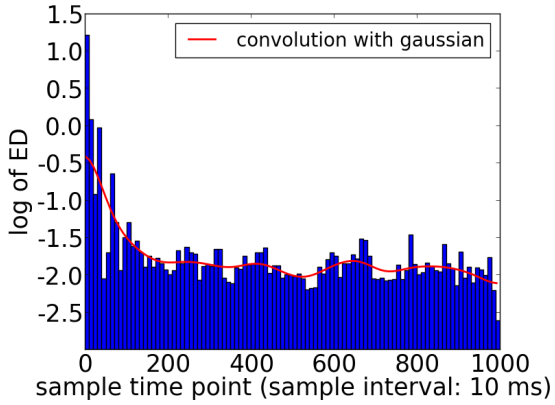
In the experiment, different random seeds for the Poisson input noise have been used for different neurons, as well as for the same neuron in different simulations. The reasons for these requirements are quite obvious: the former guarantees zero noise correlations, while the latter ensures true randomness of different trials.



(a) sample time points: 10 ms and 20 ms



(b) sample time points: 100 ms and 200 ms



(c) EDs as a function of time

Figure 3.13.: (a), (b): Comparison of firing probability distributions at different time points with the benchmark (at 1000 ms). Early time points show a different range of firing probabilities compared to the benchmark, while times points at later times show the same range of firing probabilities with the benchmark. (c): The log of EDs of different sample time points. The profile converges to a certain value after about 200 ms.

We initialized the LIF-based BM with parameters translated from the AMN weights and biases used in section 3.2.1, and the result after 5000 simulations can be seen in Fig.

3.13. Figure (3.13c) shows that the log of EDs decrease and converge to a certain value after about 200 ms.

Ideally, longer decorrelation times are better choices but are also time-consuming. In practice, we chose 200 ms as the decorrelation time for our task.

An initialization of the LIF sampler network is needed every time we start a simulation, which includes translating the theoretical BM parameters to LIF neuron parameters and building up synaptic connections between neurons. Since the network parameters are fixed during one simulation, we need to reinitialize the network every time after we update the model parameters. Also, the seed for generating random Poisson noise is changed for each neuron every time we start a new simulation during the learning process.

For the current code, one initialization of a LIF neuron network consisting of 217 neurons takes about 70 seconds (with PyNN & NEURON on IGNATZ). Hence, training the network for 10^5 times with a constant learning rate as performed in the AMN experiment would take almost 100 days. A solution to this would be initializing the LIF-network with the already trained model parameters from an AMN-based BM and train the LIF-network further for a short time with a constant small learning rate as a fine tuning.

We took the trained AMN-based BM model parameters used in section 3.2.1 as the initial parameters for the LIF-based BM and implemented a CD learning algorithm. The detailed learning process is as follows:

1. Generate a posterior sample in the same way as in section 3.1.2.
2. Initialize the LIF-based BM with model parameters (weights and biases) from the AMN domain.
3. Initialize the membrane potential of the LIF neurons according to their posterior state.
4. Run the simulation for the chosen decorrelation time and record the prior estimate.
5. The state of a neuron was set to 1 for a time interval from t to $t + \tau_{\text{ref}}$ if the neuron spikes at t, and the state after the decorrelation time was taken as the prior state of the neuron.
6. Update model parameter according to Eq. 3.2.

In step 3, the membrane potential of a neuron is set to the threshold if its state is 1, otherwise is set to a value drawn from a Gaussian distribution $N(u^{'}, \Sigma^2)$, where $u^{'}$ is a sum of the neuron's mean effective membrane potential μ (see Eq. 2.23), its bias and the membrane potential caused by recurrent input, whereas Σ^2 is the variance of u_{eff} (see Eq. 2.24). In practice, we draw 1000 values from the distribution and pick the first one which is below the threshold. If all the drawn values are above the threshold (rarely), we then set it to a value 0.001 mV below the threshold.

The result after training a LIF-based BM with noised neuron parameters for 10^4 times with $\eta = 10^{-4}$ is shown in figure 3.14.

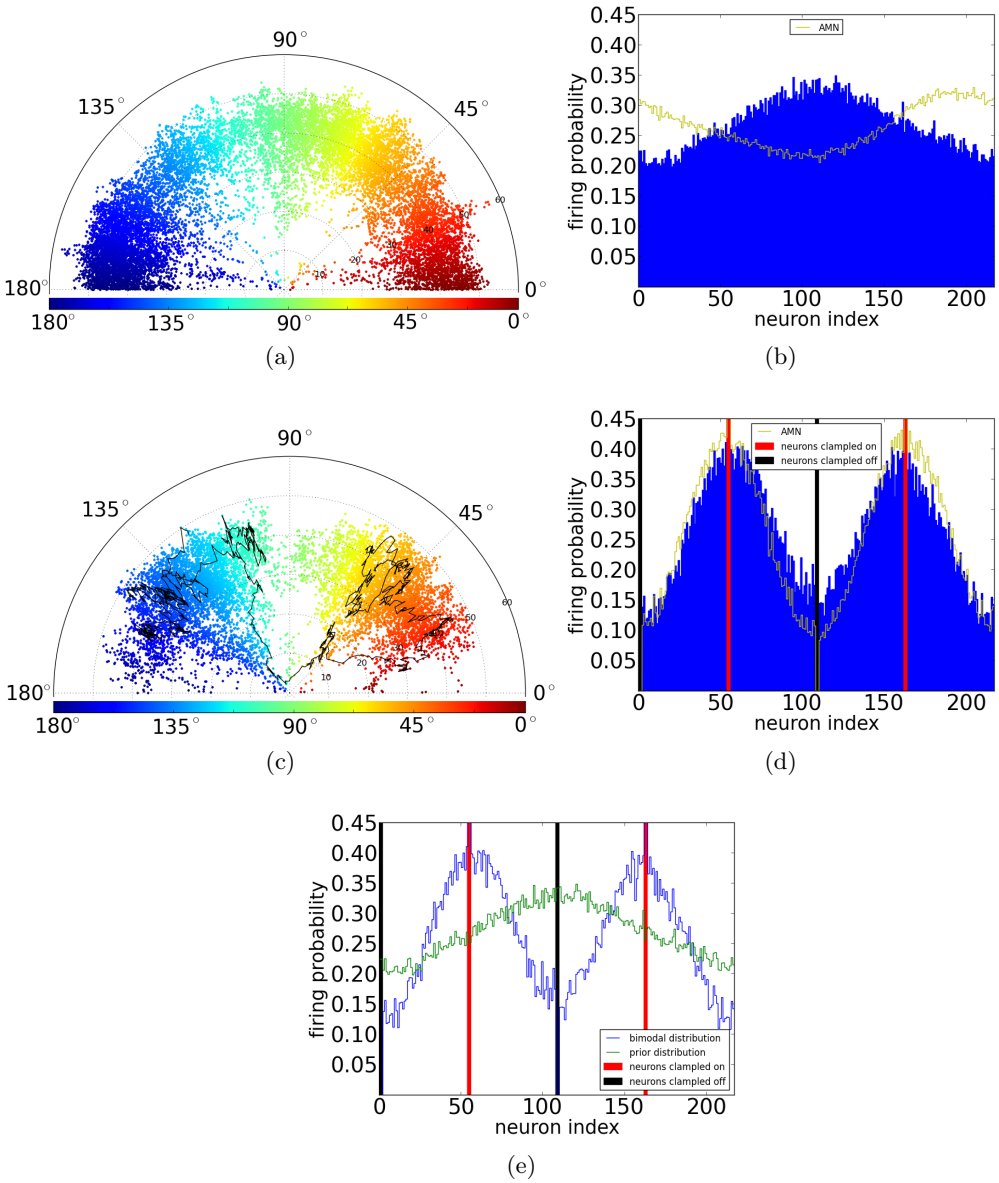


Figure 3.14.: Results after training the LIF-based BM for 10^4 times with a constant learning rate $\eta = 10^{-4}$.
(a): PV plot of prior states. The model exhibits coherent states. $T_{sim} = 2 \cdot 10^4$ ms. (b): The marginal firing distribution of prior states exhibits a functional change after a direct training. $T_{sim} = 7.5 \cdot 10^5$ ms. (c): PV plot of clamped states. $T_{sim} = 2 \cdot 10^4$, the black line shows the evolution of the network state for 1000 ms around a perceptual switch. (d): MFD of clamped states. The two peaks are almost as high as the displayed AMN curve. $T_{sim} = 7.5 \cdot 10^5$ ms. (e): Comparison of two MFDs. The central valley of the bimodal distribution is slightly higher compared to the edges due to the relatively high fire probabilities of the central neurons as exhibited in the prior states distribution.

The comparison between the prior distribution of an untrained network and a trained network can be seen in Fig. (3.15a). We can see in Fig. (3.15b) that the neurons near the clamped orientations fire with higher probabilities compared to the LIF-based BM with direct model parameter translation of noised neuron parameters. The distribution has distinctive peaks, close to the ones of the benchmark distribution. This proves the effectivity of applying a CD learning algorithm to the LIF-based BM.

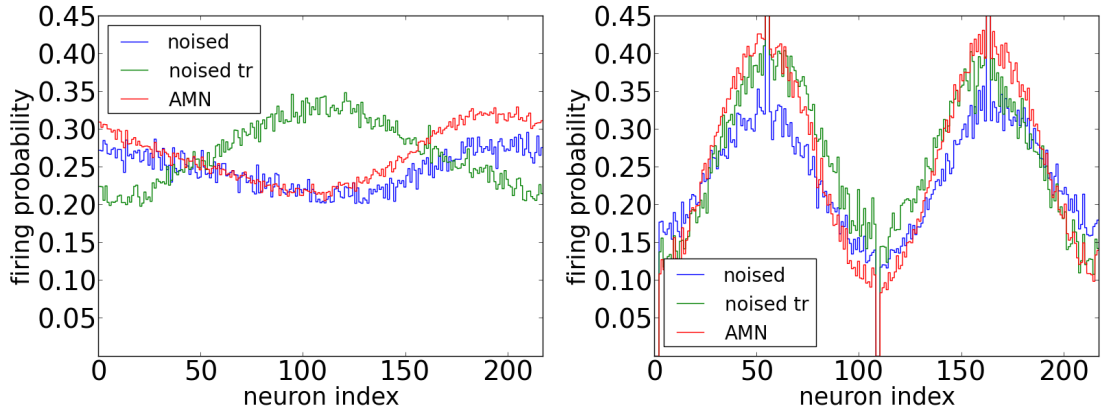


Figure 3.15.: Comparison of firing probability distributions: LIF-based BM with noised neuron parameters with direct model parameter translation (noised), LIF-based BM with noised neuron parameters with training (noised tr), and the their benchmark obtained from the AMN implementation (AMN). (a): MFDs of prior states. Due to the training, the parameters of the LIF-based BM are altered and thereby causes a functional change to the MFD of the prior states. (b): MFDs of clamped states. The training improves the performance of the LIF network significantly.

Due to the initialization time mentioned before, the training of the network cost almost 10 days. As training results would obviously benefit from longer training time, future modifications of the LIF-based BM initializations are expected to greatly improve the learning speed.

In conclusion, in this chapter we first modeled perceptual multistability with an AMN-based BM, then we translated the trained model parameters to an LIF-based BM and the result proved that an LIF-based BM is able to sample from a well-defined probability distribution. Additionally, by adding noise to the neuron parameters we showed the potential of the implementation of an LIF-based BM on a neuromorphic hardware device.

The difference of results between the two BMs (LIF-based and AMN-based) revealed the imperfection of direct model parameters translation. A careful analysis of the dynamics of a LIF sampler embedded in a larger network led to the design of a weight-rescale

method in order to reduce the deviation of the LIF-network from its benchmark.

Finally, a training algorithm for the LIF-based BM was proposed and its feasibility was demonstrated. The implementation of CD for the LIF-based BMs lays the foundation for more complex learning algorithms, which will be discussed in the next chapter for LIF-based restricted BMs.

4. Learning MNIST digits with deep learning architectures

Depending on their size and structure, restricted Boltzmann machines (RBMs) are able to learn highly complex patterns, such as visual representations of objects or speech words. They are also the building block of the multi-layer learning networks called deep Boltzmann machines, which we will introduce in the next chapter.

In the previous chapter, we implemented the contrastive divergence (CD) algorithm on an LIF-based fully visible BM. Taking this one step further leads us to the question whether training an LIF-based restricted Boltzmann machine (RBM) with hidden units is possible. We will explore this question in this chapter.

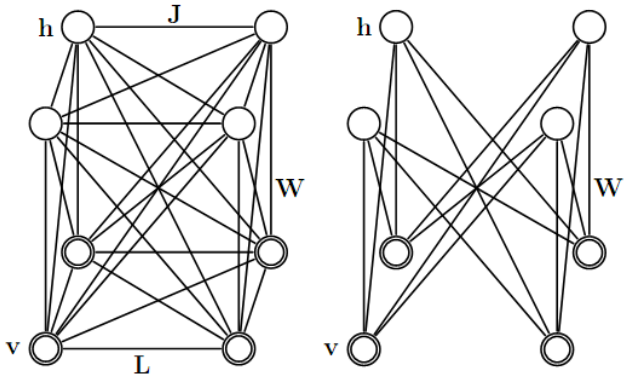
First, we will introduce the basic concept of the RBM. Afterwards, a series of learning problems and their solutions will be discussed, leading to an efficient learning algorithm, namely coupled adaptive simulated tempering (CAST).

Then, an AMN-based CAST algorithm will be developed, and its performance assessed by training an AMN-based RBM to learn MNIST handwritten digits. Based on the previous study, an LIF-based CAST algorithm will be explored. Its performance on an LIF-based RBM will be evaluated by training the model with the same task as the AMN-based RBM. The training results demonstrate the capability of the LIF-based RBM to perform unsupervised learning. The results also reveal an interesting phenomenon concerning a unique feature of LIF sampling networks.

At the end of this chapter, we will have a brief discussion on deep Boltzmann machines, which are the main focus of our future research.

4.1. Restricted Boltzmann machines

A general BM can be partitioned into a set of visible and one of hidden units (see Fig. 4.1a). A Boltzmann machine without hidden-to-hidden connections and without visible-to-visible connections is called a restricted Boltzmann machine (RBM) (see Fig. 4.1b) due to its restricted connections. Because of this connection restriction, the hidden units can be viewed as a second layer above the visible layer, which gives an RBM a concept of 'depth', making it a precursor (or building block) of multilayer deep architectures (section 4.6). The visible layer receives training data and the hidden layer learns to model dependencies between the visible units. Hidden units in a way act



(a) General BM (b) RBM

Figure 4.1.: Setting interaction terms L and J to zero obtains a RBM, (Image is taken from *Salakhutdinov and Hinton*, 2009).

as feature detectors, and because of this, are able to capture complex features of a pattern.

The connectivity reduction also allows for more efficient learning algorithms than the ones used for general BMs. The energy of an RBM is defined as

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{i,j} W_{ij} v_i h_j \ , \tag{4.1}$$

and the probability of a visible state in an RBM is defined in the same way as Eq. 2.8, which is

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})) \ . \tag{4.2}$$

4.2. Learning algorithms for RBMs

A learning algorithm can be derived, according to section 2.2.1, by differentiating $p(\mathbf{v})$ over W_{ij}, which yields

$$\frac{\partial \log p(\mathbf{v})}{\partial W_{ij}} = \langle\, v_i h_j \,\rangle_{\text{data}} - \langle\, v_i h_j \,\rangle_{\text{model}} \ . \tag{4.3}$$

This leads to a gradient ascent learning rule (*Hinton*, 2010) similar to Eq. 2.12 and 2.13

$$\Delta W_{ij} = \eta(\langle\, v_i h_j \,\rangle_{\text{data}} - \langle\, v_i h_j \,\rangle_{\text{model}}) \tag{4.4}$$
$$\Delta a_i = \eta(\langle\, v_i \,\rangle_{\text{data}} - \langle\, v_i \,\rangle_{\text{model}}) \tag{4.5}$$
$$\Delta b_j = \eta(\langle\, h_j \,\rangle_{\text{data}} - \langle\, h_j \,\rangle_{\text{model}}) \ , \tag{4.6}$$

where η represents some previously defined learning rate. However, different from a fully visible BM where $\langle z_i z_j \rangle_{\text{data}}$ can be given by the training data (each binary unit is set to 1 with a probability derived from the pixel value of the training image), the binary state of a hidden unit h_j in $\langle v_i h_j \rangle_{\text{data}}$ is obtained by

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i v_i W_{ij}) \tag{4.7}$$

given a training image for visible state $\mathbf{v}$, where $\sigma(x)$ is the logistic function $[1 + \exp(-x)]^{-1}$. $v_i h_j$ can then be taken as a data sample. Because there are no direct connections between visible units, an unbiased sample of the state of a visible unit can be obtained in a similar way given a hidden vector

$$p(v_i = 1|\mathbf{h}) = \sigma(a_i + \sum_j h_j W_{ij}) \, . \tag{4.8}$$

An exact computation of $\langle v_i h_j \rangle_{\text{model}}$ is not practical for the same reason as stated in section 2.2.1. An alternative is using CD (*Hinton*, 2002) which draws an approximate sample of $\langle v_i h_j \rangle_{\text{model}}$. One training step can be described as follows:

1. Setting the visible states according to the training data.
2. Then the hidden states are computed in parallel using Eq. 4.7, thereby obtaining a sample of $\langle v_i h_j \rangle_{\text{data}}$
3. With the hidden states, a so-called 'reconstruction' is produced by setting the state of each visible unit using Eq. 4.8, and $\langle v_i h_j \rangle_{\text{recon}}$ can be used to replace the second term in the learning rule given by Eq. 4.4.

During GS, the firing probabilities $p_i = p(v_i = 1)$ and $p_j = p(h_j = 1)$ are computed when updating the states of the units. They can be used directly in the parameter updates, which provides a much better approximation of the $\langle v_i h_j \rangle_{\text{model}}$ than the binary states (more discussions can be found in (*Hinton*, 2010)). The change of the weights and biases is then given by

$$\Delta W_{ij} = \eta(\langle p_i p_j \rangle_{\text{data}} - \langle p_i p_j \rangle_{\text{recon}}) \tag{4.9}$$
$$\Delta a_i = \eta(\langle p_i \rangle_{\text{data}} - \langle p_i \rangle_{\text{recon}}) \tag{4.10}$$
$$\Delta b_j = \eta(\langle p_j \rangle_{\text{data}} - \langle p_j \rangle_{\text{recon}}) \, . \tag{4.11}$$

During training, the probabilities of visible units can be directly taken as the pixel values of the training image.

Learning algorithms for RBMs have been discussed extensively in literature (*Hinton*, 2002), (*Tieleman*, 2008), (*Salakhutdinov*, 2010), etc. The learning efficiency of an RBM varies for different learning algorithms, and the difference in learning efficiency becomes significant with increasing complexity of the training data. In the following, we will introduce a series of learning algorithms which in the end will be combined to form an efficient learning algorithm for the application to LIF-based RBMs.

4.2.1. CD_1

CD_1 corresponds to running the model for one step of Gibbs sampling (GS) to obtain the prior state. This is fast and a reasonable approximation to the log probability gradient (see Eq. 4.3), but it will deviate significantly from the log probability gradient when the mixing rate is low (*Tieleman*, 2008). The mixing rate, briefly speaking, represents the rate at which the model can sample from different regions of the energy landscape.

RBMs will learn better if more steps of GS are used before obtaining the sample of $\langle v_i h_j \rangle_{\text{recon}}$, and CD_n is used to denote learning using n steps of GS. If enough running time is available, CD_n for greater n is preferred over CD_1.

4.2.2. Persistent contrastive divergence

Proposed by Tieleman (*Tieleman*, 2008), persistent contrastive divergence (PCD) is better at approximating the log probability gradient than CD_1 when the mixing rate is low.

In CD_n, within each learning step, we initialize the model state with a training sample and let a Markov chain propagate for n steps by GS to obtain a prior sample as an approximation of $\langle v_i h_j \rangle_{\text{model}}$. The more steps we run the more likely we will get an accurate approximation.

When the learning rate is small, the model changes only slightly between parameter updates. PCD takes advantage of this by initializing a training step at the state in which the previous step ended. Even though the model has changed slightly due to these parameter updates, this initialization can still be very close to the model distribution. In this way one keeps a 'persistent' Markov chain and is able to obtain better approximations of $\langle v_i h_j \rangle_{\text{model}}$. A detailed diagram is shown below, with CD_1 and CD_n in comparison.

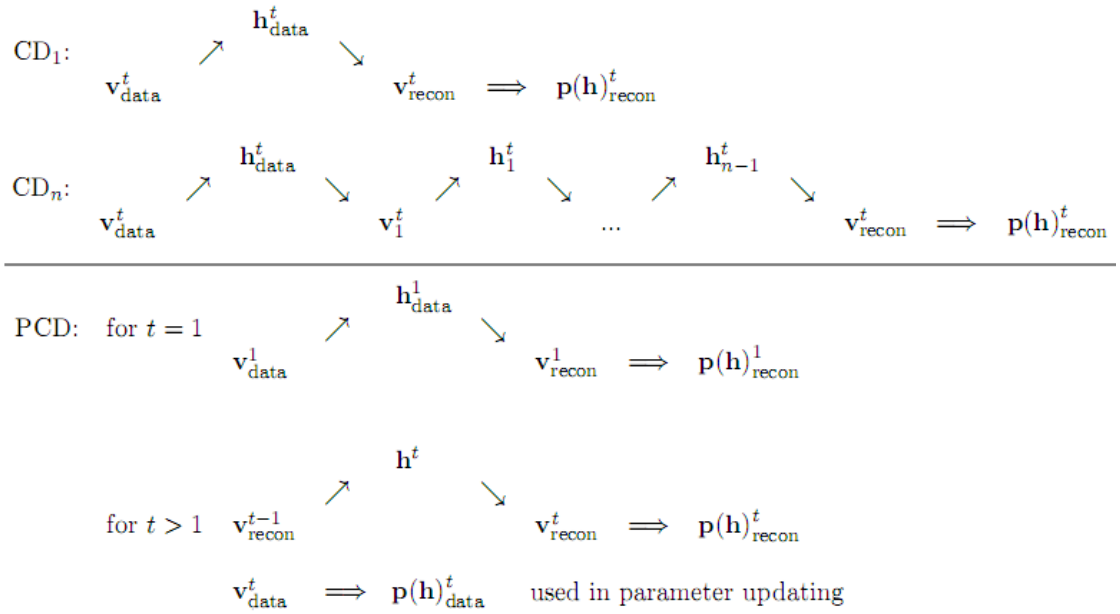


Figure 4.2.: Comparison between CD_1, CD_n and PCD. At learning step t, for CD_1, the Markov chain is initialized with $\mathbf{v}^t_{\text{data}}$ which is drawn from the training dataset. "$\longrightarrow$" denotes a step of GS updating the states of all visible or hidden units, and "$\Longrightarrow$" denotes calculating the firing probability without updating the states of the units. The $\mathbf{v}^t_{\text{recon}}$ of CD_n is obtained after n steps of GS start from $\mathbf{h}^t_{\text{data}}$. For PCD, the Markov chain is initialized with the $\mathbf{v}^{t-1}_{\text{recon}}$ obtained from the previous learning step. The $\mathbf{p}^t_{\text{data}}$ used in parameter updating is calculated separately.

Technically speaking, compared to CD_1, PCD only changes the way of obtaining $\mathbf{v}^t_{\text{recon}}$. In the last update of the PCD, the conditional distribution is calculated without updating the hidden states, since no random variable depends on the choice of the states of the hidden units. The parameters are updated immediately after $\langle\, p_i p_j \,\rangle_{\text{recon}}$ is obtained. The update rule is the same as CD in Eq. 4.9 - 4.11.

$$\Delta W_{ij} = \eta(\langle\, p_i p_j \,\rangle_{\text{data}} - \langle\, p_i p_j \,\rangle_{\text{recon}}) \tag{4.12}$$
$$\Delta a_i = \eta(\langle\, p_i \,\rangle_{\text{data}} - \langle\, p_i \,\rangle_{\text{recon}}) \tag{4.13}$$
$$\Delta b_j = \eta(\langle\, p_j \,\rangle_{\text{data}} - \langle\, p_j \,\rangle_{\text{recon}})\ . \tag{4.14}$$

However, since $\mathbf{h}^t_{\text{data}}$ is no longer determined by the Markov chain, $\mathbf{p}^t_{\text{data}}$ needs to be calculated separately using Eq. 4.7 conditioned on the states of $\mathbf{v}^t_{\text{data}}$, as shown in figure 4.2.

One thing to notice is that the persistent chain is still only making an approximation of the model distribution, because the model does change slightly during parameter updates. In the limit of infinitesimally small learning rates, the approximation will be exact and in general PCD works best with small learning rates. However, small learning

rates reduce the changes made to the energy landscape, so an insufficiently mixing persistent chain will adjust only slowly over time. This will prolong the dwell times in local minima of the energy landscape, ultimately leading to poor generative models (*Salakhutdinov*, 2010).

4.2.3. Adaptive simulated tempering

Proposed by Salakhutdinov (*Salakhutdinov*, 2010), the mixing rate of the model can be improved by performing adaptive simulated tempering (AST) instead of plain GS when updating the model state. AST is a combination of the Wang-Landau (WL) algorithm and simulated tempering (ST). We will first introduce the WL algorithm and ST separately and then show how they are combined to form the AST.

Wang-Landau algorithm

Assume the probability distribution, given by $p(\mathbf{x};\theta) = \frac{1}{Z}\exp(-E(\mathbf{x};\theta))$, is defined over the state space X. X can be partitioned into K disjoint sets $\{X_k\}_{k=1}^{K}$. The goal is to construct a Markov chain that will spend an equal amount of time in each partition and where the state updating within each partition is carried out by GS (see Fig. 4.3).

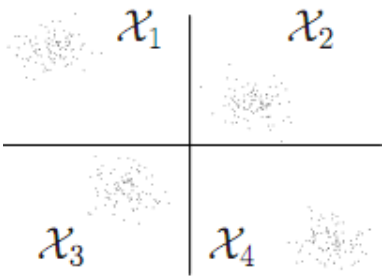


Figure 4.3.: An example of the partition of state space. The goal of the WL algorithm is to construct a Markov chain that will spend an equal amount of time in each partition. (Image is taken from *Wang and Landau*, 2001).

The WL algorithm (*Wang and Landau*, 2001) achieves this by estimating a set of factors that would properly adjust the probability in each partition. When the network remains in one partition, the corresponding adjusting factor will increase and thereby reduce the probability mass of the entire partition. This ensures that the chain spends the same amount of time in each set X_k. The algorithm proceeds as follows:

1. Let $\mathbf{g}^t$ be a vector of length K which contains the adjusting factors for each partition space. At time $t = 0$, all elements of $\mathbf{g}^t$ are initialized to 1.

2. Given a model state $\mathbf{x}^t$, sample a new state $\mathbf{x}^{t+1}$ using GS, with its invariant

probability distribution:

$$p(\mathbf{x};\theta,\mathbf{g}^t) \propto \sum_{k=1}^{K} \frac{p(\mathbf{x};\theta)}{g_k^t} I(\mathbf{x} \in X_k). \tag{4.15}$$

3. Update the adjusting factor:

$$g_k^{t+1} = g_k^t \left[1 + \gamma_t I(\mathbf{x}^{t+1} \in X_k)\right]. \tag{4.16}$$

I is the indicator function ($I(x) = 1$ if x is true, otherwise 0), and $\gamma_t > 0$ is an adapting factor as a function of time. As the chain stays in the set X_k, the adjusting factor g_k will increase, exponentially increasing the probability of moving out of X_k. As $t \to \infty$ and $\gamma_t \to 0$ for all $k \in \{1, 2, ..., K\}$ there will be convergence in probability:

$$\frac{g_k^t}{\sum_i g_i^t} \to p(\mathbf{x} \in X_k). \tag{4.17}$$

Above equation indicates that the sequence of adjusting factors will asymptotically converge to the optimal value, such that the Markov chain will spend an equal amount of time in each partition.

However, one problem of the WL algorithm is that, in practice, it is often difficult to choose a good partition of the state space X at the beginning. This problem can be solved when combining the WL algorithm with simulated tempering (ST), where appropriate partitions are defined naturally.

Simulated tempering

Simulated tempering (ST) (*Marinari and Parisi*, 1992) is an MCMC algorithm that samples from the joint distribution

$$p(\mathbf{x}, k) \propto c_k \exp\left[-\beta_k E(\mathbf{x})\right] \tag{4.18}$$

where c_k are constants, and $0 < \beta_K < \beta_{K-1} < ... < \beta_1 = 1$ are similar to the $1/k_B T$ 'inverse temperatures' as common in thermodynamics. Conditioned on k, the distribution for $\mathbf{x}$ takes the form:

$$p(\mathbf{x}|k) = \frac{1}{Z_k} \exp\left[-\beta_k E(\mathbf{x})\right]. \tag{4.19}$$

A sample from the target distribution $p(\mathbf{x})$ can therefore be obtained by simulating a Markov chain from the joint distribution $p(\mathbf{x}, k)$, and only accept the states if $k = 1$. Sampling from the joint distribution $p(\mathbf{x}, k)$ is performed by iterating through two transition operators alternately. First, according to $p(\mathbf{x}|k)$, the state $\mathbf{x}$ is updated by GS. Then, conditioned on $\mathbf{x}$, k is sampled using the Metropolis update rule with a proposal

distribution: for $2 \leq k \leq K-1$, $q(k+1|k) = q(k-1|k) = \frac{1}{2}$; and $q(2|1) = q(K-1|K) = 1$.

In ST, the high-temperature distributions facilitate mixing between many local modes. The idea is similar to increasing the temperature of a thermodynamic system, which increases the transition rates between different modes. It is noted that the Markov chain facilitates mixing when it spends an equal amount of time at each 'temperature' value. This can be achieved by setting c_k to be proportional to $1/Z_k$, which is computationally intractable. This problem is solved through a combination with the WL algorithm.

Adaptive simulated tempering

The WL algorithm solves the c_k choosing problem in ST by partitioning the state space of ST $\cup_{\{k=1\}}^{K} k \times X$ into K sets $\{k\} \cup X$, each corresponding to a different temperature value and assigned with an adaptive adjusting factor g_k. When the transition into a different partition (or temperature value) is rejected, the adaptive adjusting factor for the current partition will increase, thus increasing the probability of leaving the current partition in the next time step.

This way, one can formulate the so-called adaptive simulated tempering (AST) algorithm by combining the WL and ST algorithms. The detailed algorithm works as follows:

Algorithm: AST

Initialize: Given the adaptive adjusting factors $\{\mathbf{g}_k\}_{k=1}^{K}$ and the initial model state $\mathbf{x}^1$ at temperature 1, $k = 1$:

AST: for $n = 1 : N$ (number of iterations) do

Given $\mathbf{x}^n$, sample a new state $\mathbf{x}^{n+1}$ from $p(\mathbf{x}|k^n)$ by GS.

Given k^n, sample k^{n+1} from proposal distribution $q(k^{n+1} \leftarrow k^n)$. Accept with probability:

$$\min\left(1, \frac{p(\mathbf{x}^{n+1}, k^{n+1}) q(k^n \leftarrow k^{n+1}) g_{k^n}}{p(\mathbf{x}^{n+1}, k^n) q(k^{n+1} \leftarrow k^n) g_{k^{n+1}}}\right)$$

Update adaptive adjusting factors:

$$g_i^{n+1} = g_i^n (1 + \gamma_n I(k^{n+1} = i)),\ i = 1, ..., K.$$

end for

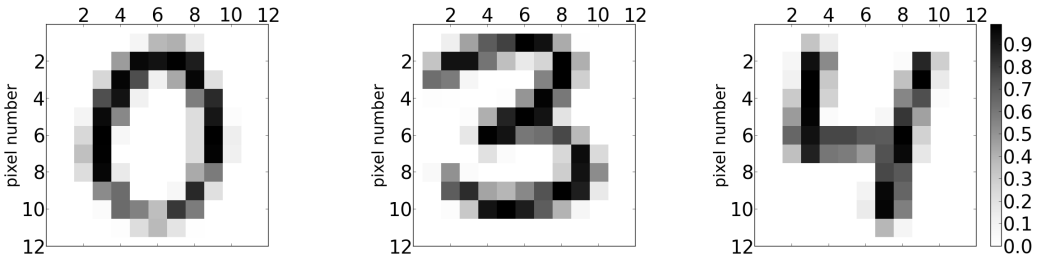
Collect data: Obtain samples from target distribution $p(x)$ by keeping $k = 1$.

The acceptance probability originates from the Metropolis update rule.

As $\gamma_n \to 0$, the ratio of adaptive adjusting factors will converge according to the probability of the Markov chain being found in a certain partition (see Eq. 4.17). This guarantees that the algorithm will roughly spend the same amount of time at each temperature value, thereby improving mixing significantly.

4.2.4. Mixing comparison

To compare the performance between AST and the original sampling method (GS), we trained an RBM with images of handwritten digits from the MNIST database (*LeCun and Cortes*, 1998). As an elementary test, we chose 3 well-recognizable images (see Fig. 4.4) corresponding to digits 0, 3 and 4. The images were reduced from 28×28 pixels to 12×12 pixels in order to downscale the network to reduce the training time.



(a) image 0 (b) image 3 (c) image 4

Figure 4.4.: 12×12 pixels images of hand written digits 0, 3, 4, reduced from the 28×28 images from the MNIST database, with pixel values ranging from 0 to 1 (corresponding to the grayscale from white to black).

The number of visible units of the RBM is set to 144 which equals the pixel number of a training image, the number of hidden units is set to 50. There are various viewpoints on how to choose the optimal number of hidden units (*Hinton*, 2010). Our experiments show that an RBM with only 3 hidden units can also learn a good representation of the training data, although it requires more training steps to achieve the same learning outcome as an RBM with 50 hidden units.

In order to reduce the computation times, we modified the parameter update rules taken from Eq. 4.12 - 4.14. In every update step, we didn't use the average of all the training data, but, only a sample of the data instead. However, during training, we iterated through all training samples (with a fixed sequence) multiple times. On average, this method is expected to converge towards the same result as the original learning rule. Since the good results obtained with this method validated our modification, we applied the same approach to all the training tasks presented here.

The RBM was trained with PCD for 10^4 times. The learning rate was chosen as exponentially decreasing for the first training steps, and then kept constant at a small

value

$$\text{for } n = 0, 1, 2...2000,\ \eta(n) = (10^{-4} - 10^{-2} + 1)^{\frac{n}{2000}} + 10^{-2} - 1\ ,$$
$$\text{for } n = 2001, ...10000,\ \eta(n) = 3 \cdot 10^{-5}\ .$$

After learning, we obtained image samples generated from the model when the network evolved freely. The pixel values of the images are represented by the firing probabilities of the visible units. We characterize a mode of the model by comparing the generated images with the training images. The generated images are subtracted from all three training images and the standard deviations (STD) of the pixel differences from the training images are calculated. More specifically, the STD is calculated as

$$\text{STD} = \left[\frac{\sum_{i=1}^{N} (P_i^g - P_i^t)^2}{N} \right]^{1/2} , \tag{4.20}$$

where N is the number of total pixels forming an image, P_i^g and P_i^t denote the ith pixel value of the generated image and training image respectively. The training image yielding the minimal STD for a point in time is assumed to be represented by the network activity. Thereby, the current network mode is identified.

We sample the prior states of the RBM using GS and AST as a comparison. The calculated STDs during the AST process is shown in Fig. 4.5.

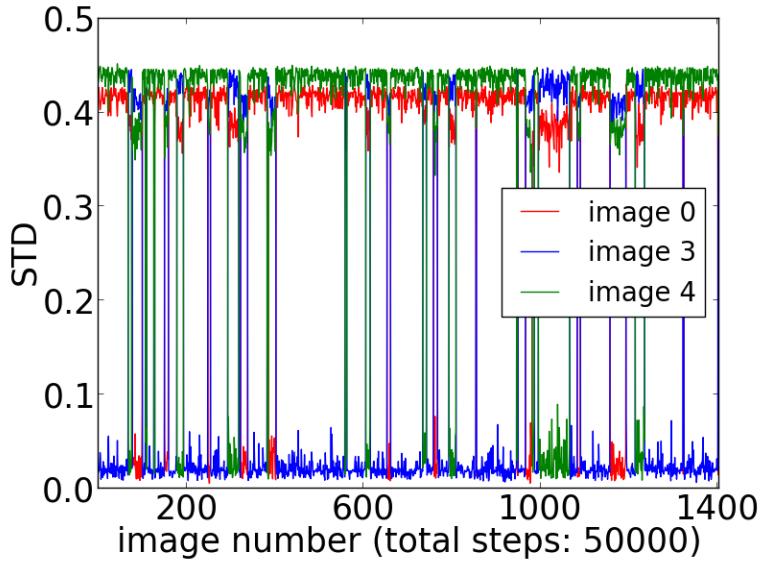


Figure 4.5.: Evolution of three STDs. The mode of the network corresponding to the image gives the smallest STD. The smallest STD values are clearly distinct from the other two. This indicates a much closer resemblance of the training image to the image generated by the network compared with the other training images. The evolution of the network modes (Fig. 4.6, bottom left) is in accordance with the variation of the smallest STDs.

The evolution of the STDs shows a large separation between values attributed to images represented by the network (low STD), and images that do not match the network dynamics at that time (high STD), indicating its feasibility as a reasonable characterization method. The results of the evolution of the modes and the consecutive samples generated from the network are shown in Fig. 4.6. Both RBMs were randomly initialized.

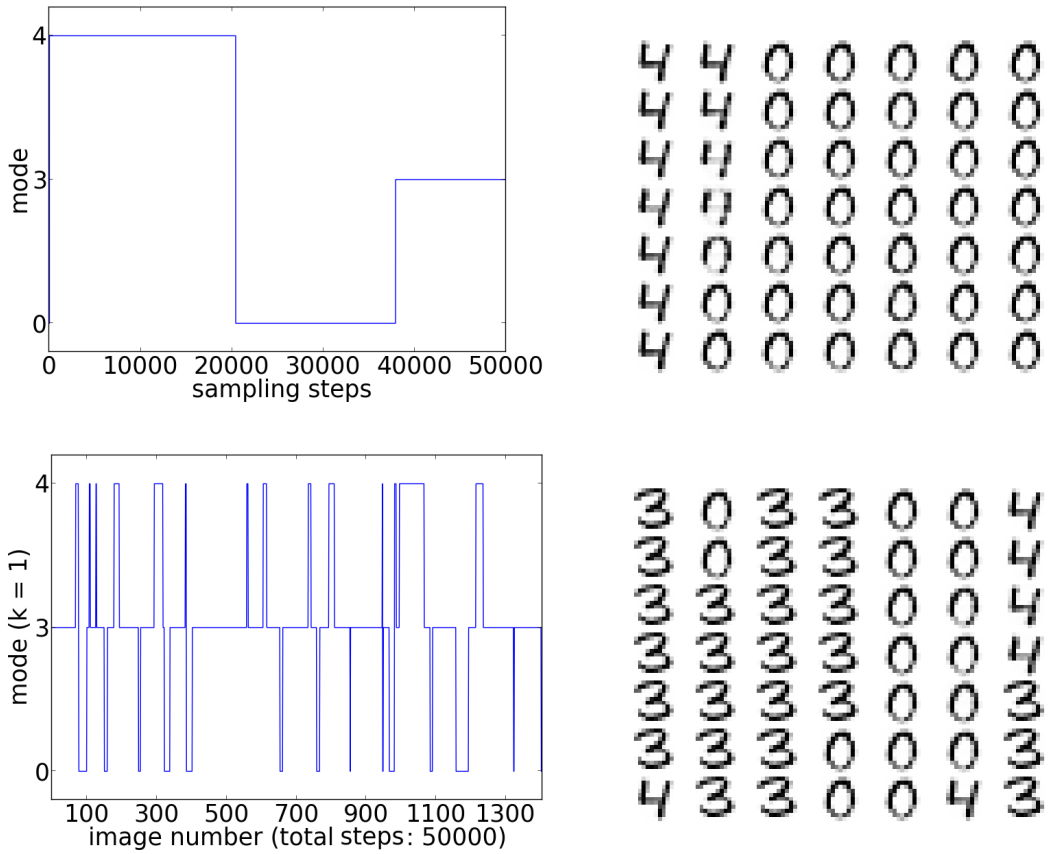


Figure 4.6.: Mixing comparison of GS and AST after learning the RBM with the PCD algorithm.
Top left: Evolution of the RBM mode obtained by GS, $N_{steps} = 5 \cdot 10^4$. It can be seen that GS mode transitions occur rarely, spending a long time in one mode. **Top right**: Consecutive images generated by the RBM (by column[1]) using GS. N_{steps}: 20140 - 20189. The RBM keeps producing the same class of images for a long time. **Bottom left**: Evolution of the RBM mode obtained by AST, recorded for temperature $k = 1$. The temperature switches to $k = 1$ for 1400 times (obtaining 1400 images) in a total of $N_{steps} = 5 \cdot 10^4$, allowing the RBM to move between different modes frequently. **Bottom right**: Consecutive images generated by the RBM using AST. Image number: 940 - 989. The RBM switches between modes more frequently, covering a large energy landscape in the same time than RBM without applied AST.

It can be easily observed that the GS mixes insufficiently and spends a long time in one mode, while AST is able to switch between different modes frequently leading to better

[1]the sequence of the images is by column, from left to right

mixing. For AST, the number of 'inverse temperatures' $\{\beta_k\}$ was set to 20, rangeing uniformly from 1 to 0.1. The adapting factor γ_t was set to 10 at a constant. Figure 4.7 shows that the adaptive adjusting factors cause the model to activate all 20 temperature values (and thereby cover their assigned partitions), forcing the RBMs to move away from the local mode.

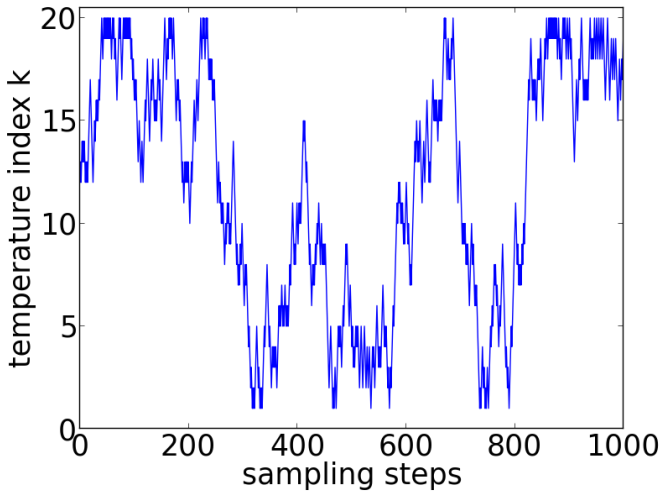


Figure 4.7.: Mixing is facilitated by the adaptive adjusting factors which cause the activation of all 20 temperature values. This implies transitions of the Markov chain to all 20 partitions of the state space, thus covering the large parts of the whole energy landscape.

4.2.5. Coupled adaptive simulated tempering

Apart from being used for sampling from the learned distribution, AST can also be applied on the learning process itself, especially when the model is trained to learn more complex patterns where effects of better mixing are more significant. When learning, the model parameters can be updated based on the states of the model when it reaches the lowest temperature (for index $k = 1$). However, performing AST between consecutive parameter updates would be computationally expensive, since it always takes some time for the model to travel (back) to the partition of the lowest temperature ($k = 1$).

In practice, we use coupled adaptive simulated tempering (CAST) (*Salakhutdinov*, 2010). There are two Markov chains in CAST. One chain, referred to as the 'slow' chain, updates the model parameters using PCD after every Gibbs update. The other chain, called the 'fast' chain, uses AST to facilitate mixing. When the fast chain reaches the state for which $k = 1$, the state is swapped with the current state of the slow chain (see Fig. 4.8).

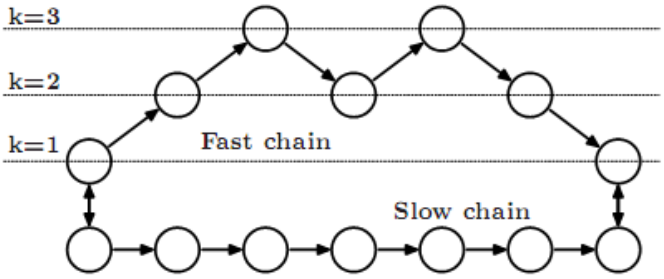


Figure 4.8.: Coupled adaptive simulated tempering (CAST). The state is swapped between the 'fast' (AST) and 'slow' (PCD) chain when the 'fast' chain reaches the state with the temperature index $k = 1$. Image taken from (*Salakhutdinov*, 2010)

The motivation behind CAST is the following: while the 'slow' chain is performing PCD, the model might easily get trapped in a local energy minimum when the learning rate is decreased to small values (for fine-tuning the model). With the aid of the 'fast' (AST) chain, the model is able to leave a local minimum more easily. This way, CAST updates the model parameters more frequently than AST and mixes better than pure PCD. Computationally, CAST is only twice as expensive as PCD.

In practice, it was suggested to swap states between the fast and slow chains after a fixed number of steps (we chose 50). More specifically, the slow chain state is swapped after Gibbs-updating the model parameters via PCD for 50 times with the most recent state of the AST chain for which $k = 1$ (after 50 Gibbs updates). This mechanism avoids the problem of continuous swapping between adjacent states as the fast chain moves around the smallest temperature value ($k = 1$).

In a brief summary, we first introduced the CD algorithm applied to RBMs. Compared to CD, PCD approximates the log probability gradient (see Eq. 4.3) more closely. However, both algorithms are based on plain GS which mixes slowly when the learning rate is small. AST facilitates mixing by defining a temperature which is adaptively changed as well. Finally, the combination of AST and PCD leads to CAST, which updates the model parameters efficiently while maintaining a good mixing. CAST is an efficient learning algorithm, especially for learning more complex patterns. In the next section we will show training results obtained from an RBM using CAST as its learning algorithm.

4.3. Training result

Before using the CAST algorithm for training the AMN-based RBM, we first trained an RBM based on binary units to test the performance of CAST in learning more complex data.

100 images of handwritten digits corresponding to 0 to 9 (10 for each class), which were taken from the MNIST database. To reduce the computation time, all images were reduced to 12×12 pixels as before (see Fig. 4.9).

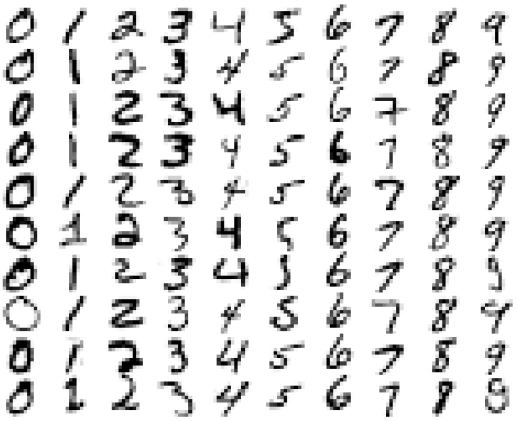


Figure 4.9.: Training data: 100 handwritten digits taken from the MNIST database and contracted to 12×12 pixels.

We set the visible units number of the RBM to 144 in accordance to the pixel number of the image, the hidden units number was set to 100. We trained the network for $2 \cdot 10^5$ parameter updates, with exponentially decreasing learning rates:

$$\begin{aligned} &\text{for } n = 0 - 1.5 \cdot 10^5, \eta(n) = (10^{-4} - 10^{-2} + 1)^{\frac{n}{1.5 \cdot 10^5}} + 10^{-2} - 1 \ , \\ &\text{for } n = 1.5 \cdot 10^5 - 2 \cdot 10^5, \eta(n) = (3 \cdot 10^{-5} - 10^{-4} + 1)^{\frac{n - 1.5 \cdot 10^5}{5 \cdot 10^4}} + 10^{-4} - 1 \ . \end{aligned} \tag{4.21}$$

After learning, the performance of the RBM was assessed by comparing the generated images (using AST) with the training images. We ran the (randomly initialized) network for a total $8 \cdot 10^4$ sampling steps and obtained 2000 images. The mode of the RBM was characterized in the same way as in section 4.2.4, with each mode corresponding to a training image, which makes the total mode number to be 100. The distributions of the generated images and the mean STDs in each mode are shown in figure 4.10, together with a series consecutive image samples generated by the RBM.

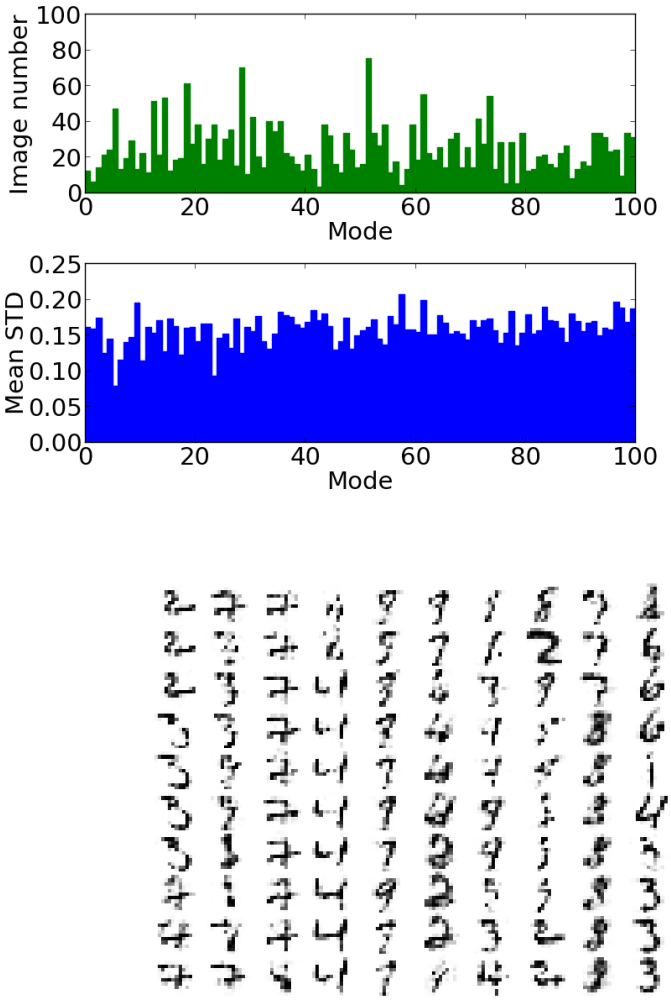


Figure 4.10.: Statistics of generated images from a simulation for $8 \cdot 10^4$ Sstps, after training the RBM to learn 100 images using the CAST algorithm.
Top: Distributions of generated images and the mean STDs in each mode. Images generated from the network cover all modes. The lower the mean STD, the higher the similarity between the generated image and the training sample. **Bottom**: A series of consecutive image samples generated from the RBM (by column). The network switches frequently between different modes (corresponding to images).

The result shows that the RBM with the CAST algorithm was able to learn all the training data and generate well recognizable image samples. More elaborate learning rate policy can be studied to improve the training result.

One thing to notice is that, due to the time limitations, the model assessment method

we used is not an elaborate evaluation of the model. Our goal is to test the capability of CAST to learn more complex training data and to explore the feasibility of its application to LIF-based RBMs. For a more strict evaluation of the model, prospective methods such as estimating the lower bound of the log probability of the training data (*Salakhutdinov and Hinton*, 2009) can be considered.

4.4. Learning MNIST handwritten digits with AMN-based RBMs

Similarly to the working protocol in the previous chapter, prior to the study of LIF-based RBMs, we will first build AMN-based RBMs as a theoretical reference.

4.4.1. AMN-based CAST

In the last section we showed the efficiency of the CAST algorithm in learning complex training data. To increase the efficiency of AMN-based learning of RBMs, our first step is to develop an AMN-based CAST algorithm. Since (*Buesing et al.*, 2011) have already proven that neural sampling with AMNs can be interpreted as MCMC sampling. The implementation of the CAST algorithm (which is originally based on GS) with AMNs is therefore assumed to be feasible.

The AMN-based CAST algorithm was essentially a combination of an AMN-based PCD and an AMN-based AST.

AMN-based PCD

For the AMN-based PCD, the probabilities $p(v_i = 1)$ and $p(h_j = 1)$ were not used to calculate the model parameter updates. Instead, we used the binary states of the AMNs. The reason for this can be found in the firing probability of an AMN. According to Eq. 2.16, it was defined as

$$\begin{aligned} p(z_k(t) = 1 | \zeta_k(t) \in \{0, 1\}, u_k(t)) &= \sigma(u_k - \log \tau) \\ &= \frac{1}{1 + \tau e^{-u_k}} \,. \end{aligned} \tag{4.22}$$

The AMN will only fire according to this probability when $\zeta = 0$ or 1. During the refractory period, the state of the AMN is independent of the firing probability. This is different from a BM based on binary units, where the firing probability determines the state of the unit at every discrete time point.

The use of binary states in parameter updating leads to a modification of the PCD chain, as shown in table 4.1:

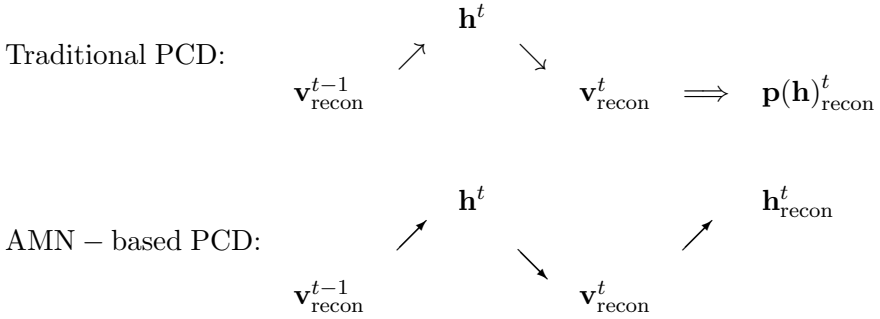


Table 4.1.: Comparison of the AMN-based PCD and the PCD applied to BMs based on binary units, at training step t.
In PCD applied on binary units, "$\rightarrow$" denotes a step of GS updating the states of all visible or hidden units, and "$\Longrightarrow$" denotes calculating the firing probability without updating the states of the units. For AMN-based PCD, "$\rightarrow\!\!\!\!\!\!\rightarrow$" denotes updating the states of all visible or hidden units over τ time steps, which is the minimum time that allows all units with $z_k = 1$ to switch to $z_k = 0$.

For the AMN-based PCD, the binary states of the hidden units $\mathbf{h}^t_{\text{recon}}$ are sampled out in the last step, since they are used for updating the model parameters. The model parameters updating rules corresponds to:

$$\Delta W_{ij} = \eta(\langle\, v_i h_j \,\rangle_{\text{data}} - \langle\, v_i h_j \,\rangle_{\text{recon}}) \tag{4.23}$$
$$\Delta a_i = \eta(\langle\, v_i \,\rangle_{\text{data}} - \langle\, v_i \,\rangle_{\text{recon}}) \tag{4.24}$$
$$\Delta b_j = \eta(\langle\, h_j \,\rangle_{\text{data}} - \langle\, h_j \,\rangle_{\text{recon}}) \,. \tag{4.25}$$

In practice, the pixel value of the training images are interpreted as firing probabilities of the visible units. By sampling from these probability distributions, binary $\mathbf{v}_{\text{data}}$ vectors can be obtained. $\mathbf{h}_{\text{data}}$ is obtained by clamping (setting the bias to an extreme value) the visible units states on $\mathbf{v}_{\text{data}}$ and running the network for a minimum 'decorrelation time'. To obtain a state for the parameter update, the network first needs to run for at least τ sampling steps in order to decorrelate $\mathbf{z}(t')$ from $\mathbf{z}(t)$ (units with $\zeta(t) = \tau$, $z(t) = 1$ require at least τ time steps before they can switch to $\zeta(t') = 0$, $z(t') = 0$). In GS, $\tau = 1$, which means that the updating speed of the AMN-based RBM is approximately τ times slower compared to the RBM based on binary units.

AMN-based AST

In AST applied to RBMs based on binary units, the inverse temperature β_k was updated once the state of the model was renewed. For the AMN-based AST, due to the same decorrelation considerations as above, one needs to wait for at least τ steps to sample a new inverse temperature from the proposal distribution. More specifically, we set the time for updating the inverse temperature to be $t = k\tau$, where k represents an integer.

4.4.2. Training result

In order to reduce computation time, we trained the model with 15 images of MNIST handwritten digits from three classes, namely 0, 3 and 4, as shown in Fig. 4.11.

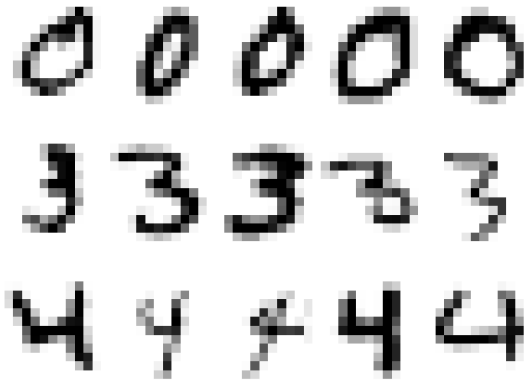


Figure 4.11.: Training images: 15 images of handwritten digits from classes 0, 3 and 4 taken from the MNIST database, reduced to 12×12 pixels.

The number of visible and hidden units were set to 144 and 100 respectively. The refractory time was chosen as $\tau = 10$. The model was trained for $3 \cdot 10^4$ parameter updates. The learning rate was chosen as an exponentially decreasing function of time with two different decay rates

$$\text{for } n = 0 - 2.2 \cdot 10^4,\ \eta(n) = (10^{-4} - 10^{-2} + 1)^{\frac{n}{2.2 \cdot 10^4}} + 10^{-2} - 1\ ,$$

$$\text{for } n = 2.2 \cdot 10^4 - 3 \cdot 10^4,\ \eta(n) = (3 \cdot 10^{-5} - 10^{-4} + 1)^{\frac{n - 2.2 \cdot 10^4}{8 \cdot 10^3}} + 10^{-4} - 1\ .$$

After learning, we ran the (randomly initialized) network for $2 \cdot 10^4$ sampling steps, during which the network generated more than 800 images. The distribution of the generated images and the mean STDs are shown in figure 4.12, together with the evolution of the network modes and a series consecutively generated image samples.

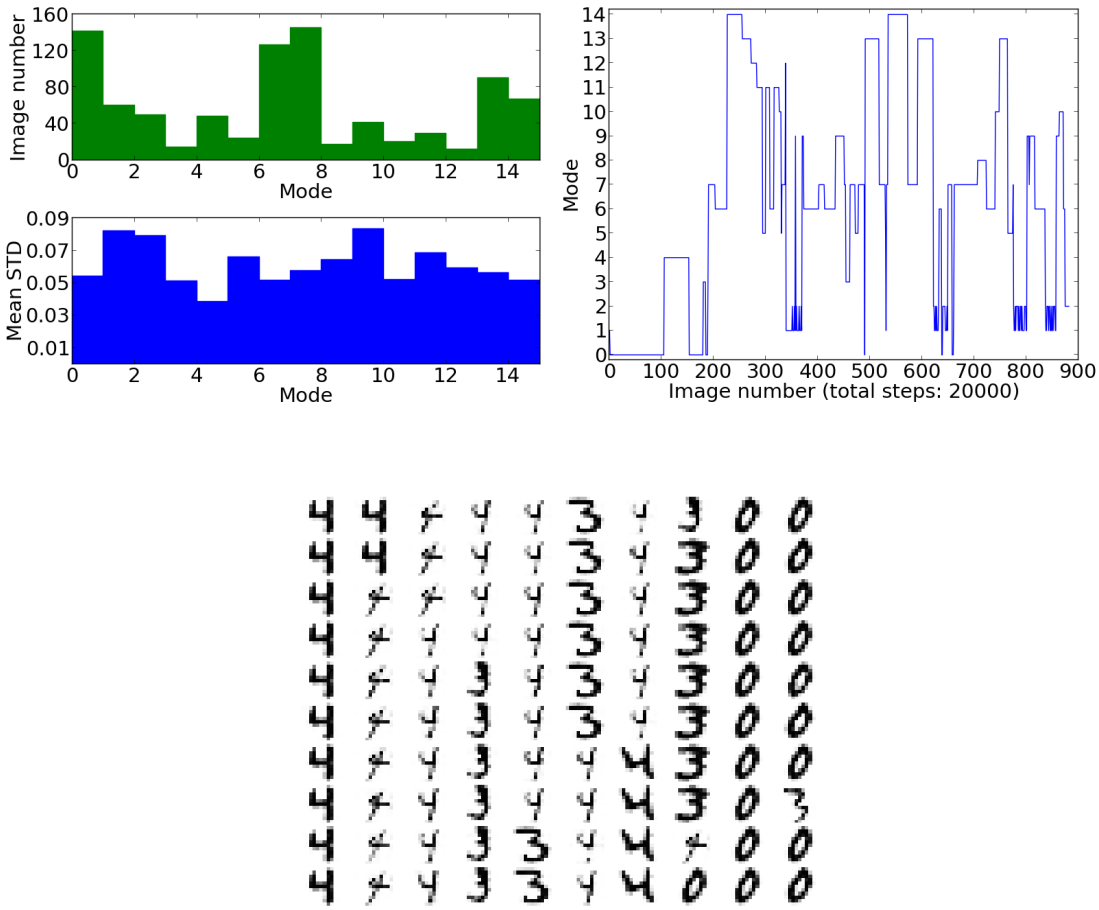


Figure 4.12.: Statistics of generated images from a simulation of $2 \cdot 10^4$ time steps, after training the AMN-based RBM with 15 images using the CAST algorithm. **Top left**: Distribution of the generated images and the mean STDs in each mode. Images generated from the network cover all modes. **Top right**: Evolution of network modes. The network is able to mix among all 15 modes. **Bottom**: A series of consecutive image samples generated from the network (by column). The network switches between different image classes frequently.

Note that the pixel values of a generated image are no longer taken as the firing probabilities of visible units (AMNs). Instead, we obtained a list of binary states of the AMNs at the end of a simulation. Then, an RBM based on binary units was initialized with the AMN model parameters and the prior to that acquired binary states of the hidden units (AMNs). The firing probabilities computed from the states of the RBMs based on binary units are then taken as the pixel values.

The results show that the AMN-based RBM with the AMN-based CAST algorithm is able to learn the training data and generate image samples covering all 15 modes. The network switches frequently between different modes. The mean STDs can be found in an interval 0.03 - 0.09, which indicates a high resemblance between the generated images and the presented training data.

However, the occurrence rates of the generated images in each mode are not evenly distributed. To improve this outcome, we increased the simulation steps to $2 \cdot 10^5$, leading to a more uniform distribution. But still distinct occurrence peaks can be seen in the resulting distribution. Even after training the RBM further for a longer period with a significantly smaller learning rate $\eta = 3 \cdot 10^{-5}$, the distribution landscape changed without a clear tendency towards uniformity. It proved to be difficult to reach a distribution which does not exhibit preferences towards certain images. One reason for this is the number of modes, which in our case is 15. For the previous simulation with 3 modes, an approximately uniform distribution is easier to achieve. The increase of network modes will obviously raise the difficulty of reaching a uniform distribution.

4.5. Learning MNIST handwritten digits with LIF-based RBMs

For the LIF-based RBMs, we chose the set of neuron parameters (conductance-based) from table 4.2.

Name	Value	Units	Description
`v_rest`	-50.0	mV	Resting membrane potential u_{rest}
`cm`	0.2	nF	Capacity of the membrane C_{m}
`tau_m`	0.1	ms	Membrane time constant τ_{m}
`tau_refrac`	10.0	ms	Duration of refractory period τ_{ref}
`tau_syn_E`	10.0	ms	Decay time of the excitatory synaptic conductance τ_{syn}
`tau_syn_I`	10.0	ms	Decay time of the inhibitory synaptic conductance τ_{syn}
`e_rev_E`	0.0	mV	Reversal potential for excitatory input $E_{\text{exc}}^{\text{rev}}$
`e_rev_I`	-100.0	mV	Reversal potential for inhibitory input $E_{\text{inh}}^{\text{rev}}$
`v_thresh`	-50.0	mV	Spike threshold ϑ
`v_reset`	-50.01	mV	Reset potential after a spike u_{reset}
`i_offset`	0.0	nA	Offset current I_{off}

Table 4.2.: Neuron parameters for software (NEURON) simulation.

Compared to the neuron parameters used in the LIF-based BM in the previous chapter, we decreased the membrane time constant τ_{m} by a factor of 10, thus increasing the leak conductance g_{l}. This reduces the ratio of the conductance of inter-neuron synapses g^{n} to the total conductance g_{tot}, thus decreasing the deviation during weight translation (see Eq. 3.4). We also reduced the duration of the refractory period and the synaptic time constant to 10 ms in order to save simulation time.

4.5.1. Learning algorithm

We implemented a CAST algorithm consisting of an LIF-based PCD and the AMN-based AST. Currently, the LIF-based AST is not implemented yet and requires a separate investigation. As concluded in the previous chapter, the critical issue is that the algorithm used for updating the parameters must be adapted to work for the LIF neural network. The application of the AMN-based AST to facilitate mixing is reasonable since the AMN-based network serves as the benchmark for the LIF-based network. The obtained results proved the feasibility of this approach.

Likewise, for the LIF-based PCD, the states of the hidden units are determined by probabilities conditioned on states of the visible units, as described in Eq. 4.7. For the PCD learning procedure, the visible unit states needed to be constant during the mixing period prior to the sampling of the hidden unit states. The states of these units were clamped by setting the mean membrane potentials to extreme values. The states of the hidden units were then read out after running the simulation for a decorrelation time of 100 ms (for the evaluation of an appropriate decorrelation time value, see section 3.2.4).

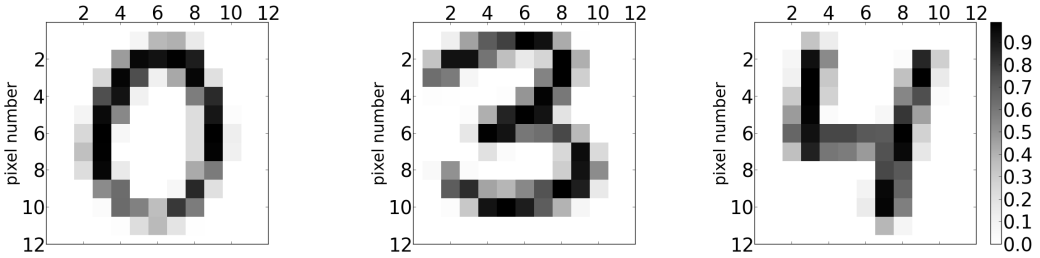
Originally, in PCD as described in section 4.2.2, in the 'reconstruction' step, the visible states $\mathbf{v}_{\text{recon}}$ should be obtained after running a simulation with clamped hidden states. However, in practice, we only initialized the membrane potential (the same method as section 3.2.4) according to the states of visible and hidden units and obtained good results with this method as well.

The states between the two chains were also interchanged after every 50 parameter updates in PCD, as described in section 4.2.5.

4.5.2. Learning 3 images of handwritten digits

For the LIF-based PCD, the network needs to be initialized three times for every step of parameter updates. These includes $\mathbf{v}_{\text{data}} \rightarrow \mathbf{h}_{\text{data}}$ for parameter updating (described by Eq. 4.23 - 4.25), $\mathbf{v}_{\text{recon}} \rightarrow \mathbf{h}$ for the 'persistent' sampling chain, and $(\mathbf{v}_{\text{recon}}, \mathbf{h}) \rightarrow (\mathbf{v}_{\text{recon}}, \mathbf{h}_{\text{recon}})$ for the 'reconstruction' step. This slows down the training of the LIF-based RBM by almost a factor of three compared to the training of the LIF-based fully visible BM, where the network only needs to be initialized once for every parameter update.

To save computation time for the first trials, we initially trained the LIF-based RBM with 3 well recognizable images, as in section 4.2.4.



(a) image 0 (b) image 3 (c) image 4

Figure 4.13.: 12×12 pixel images of handwritten digits 0, 3, 4, reduced from the 28×28 pixel images from the MNIST database, with pixel values ranging from 0 to 1 (corresponding to the grayscale from white to black).

The visible and hidden unit number of the LIF-based RBM was set to 144 and 50 respectively,and the decorrelation time of the model was evaluated and chosen to be 100 ms.

For the current LIF sampling code, the initialization of a network consisting of 194 neurons takes about 50 seconds (on IGNATZ). One full parameter update therefore required 150 seconds. This made a direct training of the model 'from scratch' (which would need more than 10^4 parameter updates, thereby costing more than 17 days) not practical.

In practice, we initialized the model with already pretrained AMN-based RBM parameters. The LIF-based RBM was trained further for $3 \cdot 10^3$ parameter updates (cost about 6 days on DOPAMINE[2]) with a constant learning rate $\eta = 10^{-4}$. After training, we ran a simulation to evaluate the model. For comparison, the simulation result of the direct parameter translation without training is shown in figure 4.14. The result after training is shown in figure 4.15.

[2]DOPAMINE: Intel(R) Core(TM) i7 CPU 920 2.67GHz

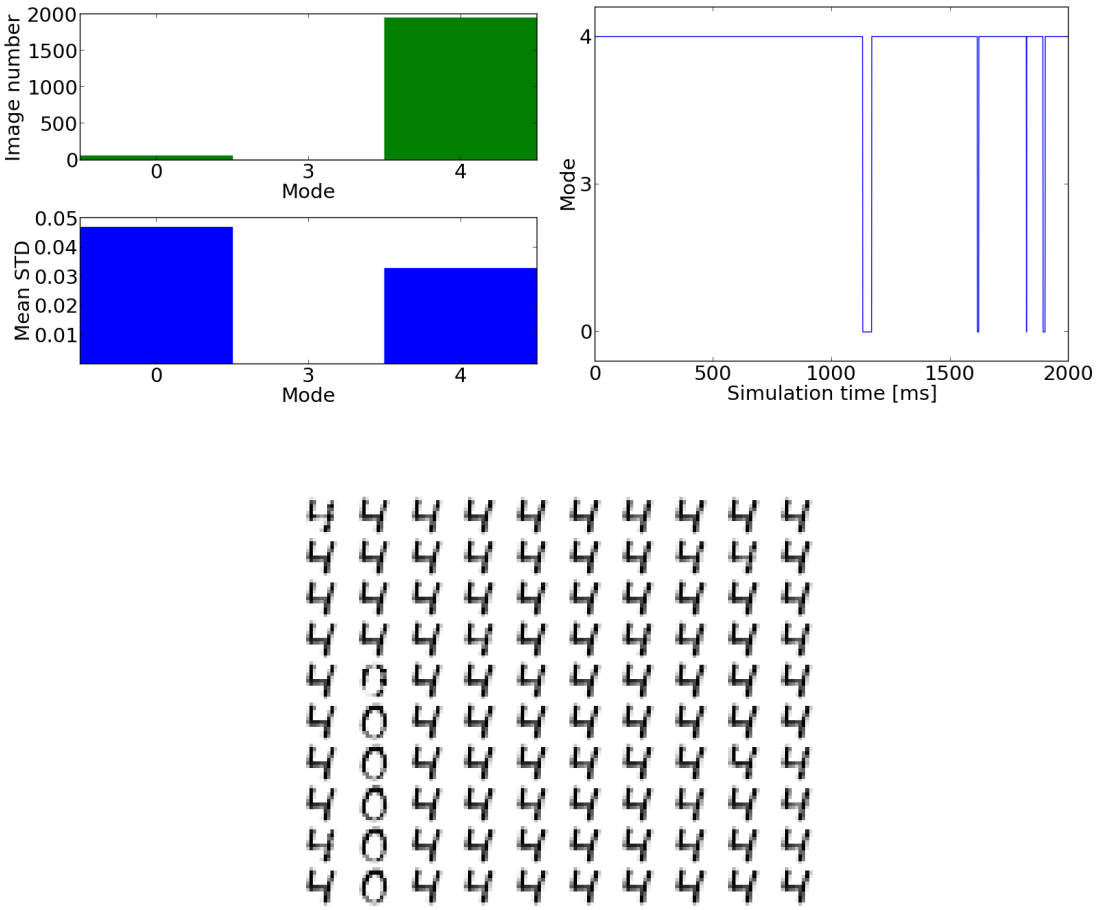


Figure 4.14.: Statistics of generated images from a simulation with $T_{sim} = 2 \cdot 10^3$ ms after direct parameter translation.
Top left: Distribution of generated images and the mean STDs in each mode. The network is almost permanently stuck in mode '4'. **Top right**: Evolution of network modes. The network stays in one mode for a long time and rarely switches to other modes. **Bottom**: A series of consecutive image samples (T_{sim}: 1600 - 1700 ms) generated by the network (by column). The network mainly produces image 4.

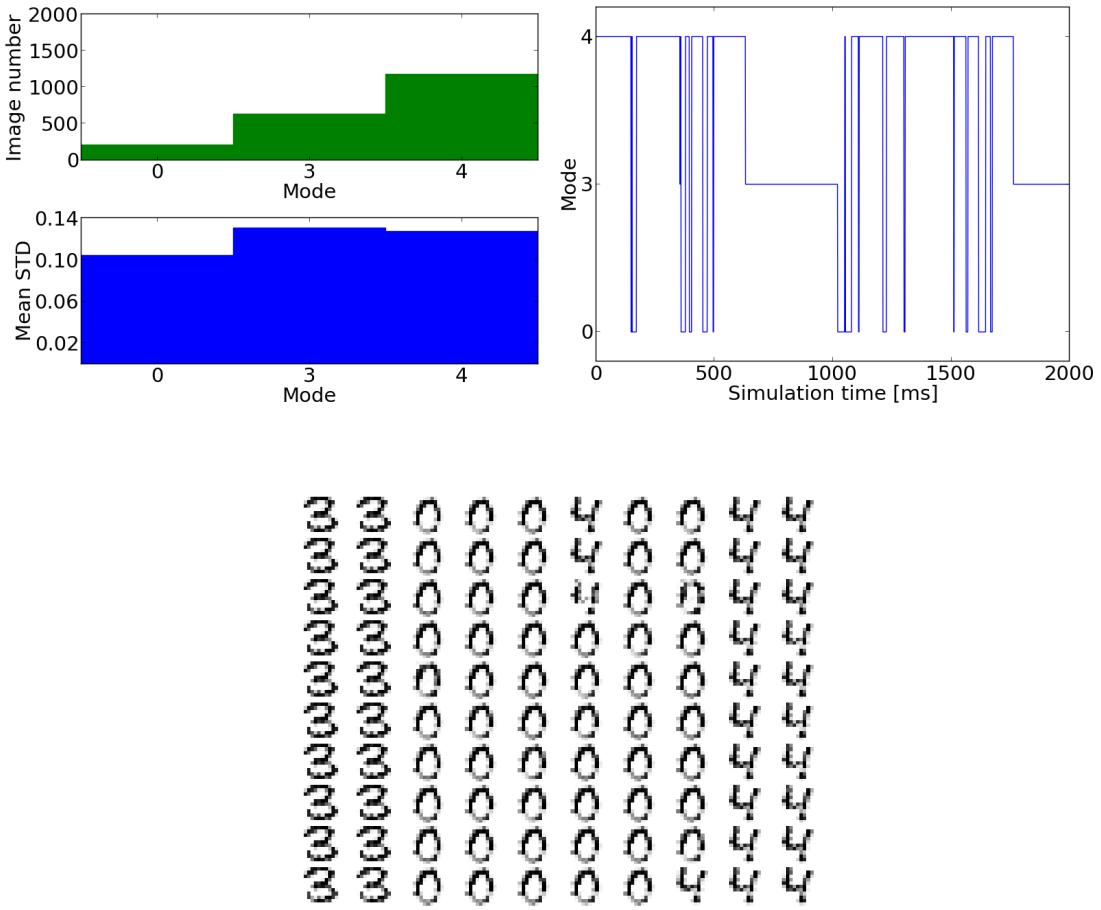


Figure 4.15.: Statistics of generated images from a simulation with $T_{sim} = 2 \cdot 10^3$ ms, after training the LIF-based RBM to learn 3 images using the CAST algorithm. **Top left**: Distribution of generated images and mean STDs for each mode. Images generated by the network cover all modes. **Top right**: Evolution of network modes. The network is able to switch between different modes frequently. **Bottom**: A series of consecutive image samples (T_{sim}: 1000 - 1100 ms) generated by the network (by column). The network switches between different image classes frequently.

The pixel values of a generated image were obtained by a similar method as in the AMN simulation. We first ran a simulation and obtained a list of binary states of the LIF neurons. Then, an RBM based on binary units was initialized with LIF model parameters (translated Boltzmann weights and biases using Eq. 2.32, 2.33) and the binary states of the hidden units (neurons). The value of a pixel point was taken as the firing probability of the corresponding visible unit of the RBM based on binary units.

The results showed that without training, the network barely mixed during simulation and basically stayed in one mode. After training using the CAST algorithm, the network was able to switch between different modes frequently.

Something worth noticing was that, after training, all results were obtained by simply running the network without any algorithms to facilitate mixing. This relates to the more interesting case when we train the model to learn 15 images in the next section.

Estimation of the training quality

To estimate the quality of the training of the network, we used a method called conservative sampling-based likelihood (CSL) estimator (*Bengio and Yao*, 2013). The CSL estimator uses a Markov chain which is defined for the model to collect samples from the generative model. For RBMs, the Markov chain alternatively samples from hidden units and visible units such that the conditional distribution $P(v|h)$ is well defined. The CSL estimate can be calculated by

$$\mathrm{CSL} = \frac{\sum_j^N \left\{ \log \left[\frac{\sum_i^M P(\mathbf{v}_j|\mathbf{h}_i)}{M} \right] \right\}}{N} , \tag{4.26}$$

where $\mathbf{v}_j$ is taken from a set containing N test samples and $\mathbf{h}_i$ is one of the M hidden states sampled from the model.

In practice, we generated 100 test samples for each training image and 1000 samples for the hidden state. A comparison of CSL estimators during the training of three networks is shown in Fig. 4.16.

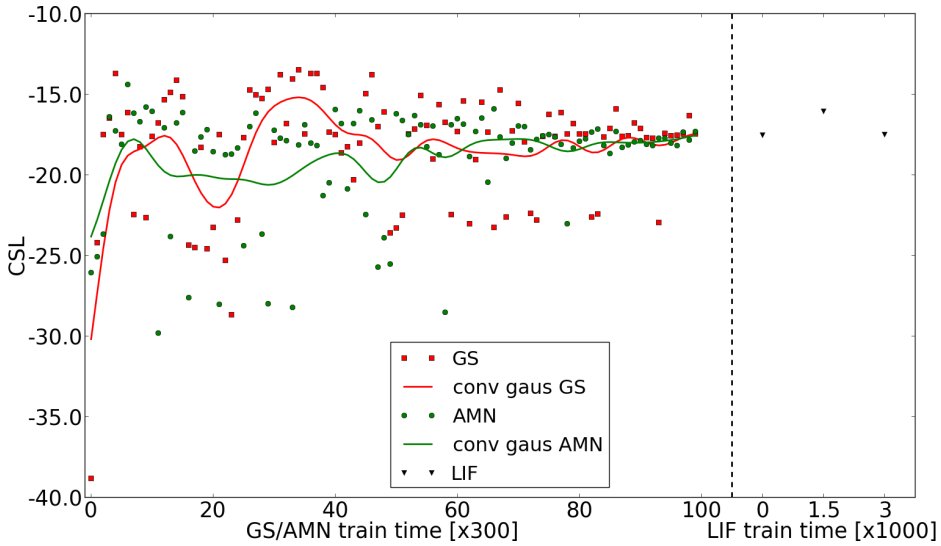


Figure 4.16.: Evolution of CSLs of the traditional GS-based RBM, the AMN-based RBM and the LIF-based RBM. Both the GS-based and AMN-based RBM were trained for $3 \cdot 10^4$ times with the same exponentially decreasing learning rates as described in Eq. 4.27. The LIF-based RBM was trained with a constant learning rate $\eta = 10^{-4}$ for $3{\cdot}10^3$ times, with initializing parameters translated from the trained model parameters of the AMN-based RBM.

The learning rates of the GS-based and AMN-based RBM are as follows:

$$\begin{aligned} &\text{for } n = 0 - 2.2 \cdot 10^4,\ \eta(n) = (10^{-4} - 10^{-2} + 1)^{\frac{n}{2.2 \cdot 10^4}} + 10^{-2} - 1\ , \\ &\text{for } n = 2.2 \cdot 10^4 - 3 \cdot 10^4,\ \eta(n) = (3 \cdot 10^{-5} - 10^{-4} + 1)^{\frac{n - 2.2 \cdot 10^4}{8 \cdot 10^3}} + 10^{-4} - 1\ . \end{aligned} \tag{4.27}$$

For both networks we collected the model parameters at 100 time points during the training. Due to time limitations, we only recorded model parameters at 3 time points for the LIF-based RBM.

The results show that for the GS-based and AMN-based RBM, the CSL rises quickly at the beginning of the training and gradually converges to a certain value with decreasing fluctuations (due to the decrease of learning rates). This accumulation point indicates the convergence of the parameters to certain target model parameters. These target model parameters are assumed to be a a global minimum for the RBM, yielding optimal results to generate the training images. This convergence point can be used evaluate the quality of our learning approach.

One particular characteristic of the results in Fig. 4.16 is the high variance of data points from the AMN-based and GS-based RBM. One reason for this variance could be that we modified the parameter update rules taken from Eq. 4.4 - 4.6 (see also section

4.2.4). In order to decrease computational resources, we do not compute the averages of the training data as described in Eq. 4.4 - 4.6. Instead, for each parameter update, we use only one training sample. This causes a more erratic evolution of the network model parameters, since the parameter updates based only one state are more susceptible to changes towards local minima. Still, in the long term evolution the parameters should converge towards the optimal values.

Other reasons for the high variance might be the insufficient number of samples of the test data and hidden states in the calculation of the CSL.

The parameters of the LIF-based RBM have been translated from the AMN domain to the LIF domain (see Eq. 2.32 - 2.33 in chapter 2) and trained for another 3000 steps, as previously described. The three resulting CSL data points in Fig. 4.16 show a good correspondence to the accumulation point resulting from the AMN-based RBM. Nevertheless, a significantly larger sample size is necessary to fully assess the quality of our LIF network training method.

In future work, we plan to evaluate various modified versions of the presented training mechanisms to optimize the convergence speed.

In the following, we assign different inference tasks to our trained LIF-based RBMs to assess their performance.

Fixed pattern completion

In general, a generative model is used (as its name already says) to generate data samples from the data it previously learned. If the model allows (partial) input, it can perform pattern completion, i.e., sample from the conditional (posterior) distribution given some (incomplete) observation.

In our particular case of binary BMs trained with grayscale images, partial observations are represented by clamping neurons in the visible layer to states determined by the observed pixel values.

For an LIF-based RBM which was trained to learn 3 images, we clamped eight feature points of image '0' to $z = 1$ and clamped two feature points of image '3' and '4' to $z = 0$ to avoid ambiguous recognition. The clamping was performed by setting the biases of the corresponding visible units to extreme values (we chose 50 and -50 for clamping to 1 and 0 respectively in the AMN domain). The result of a simulation with these settings for 10^3 ms is shown in Fig. 4.17.

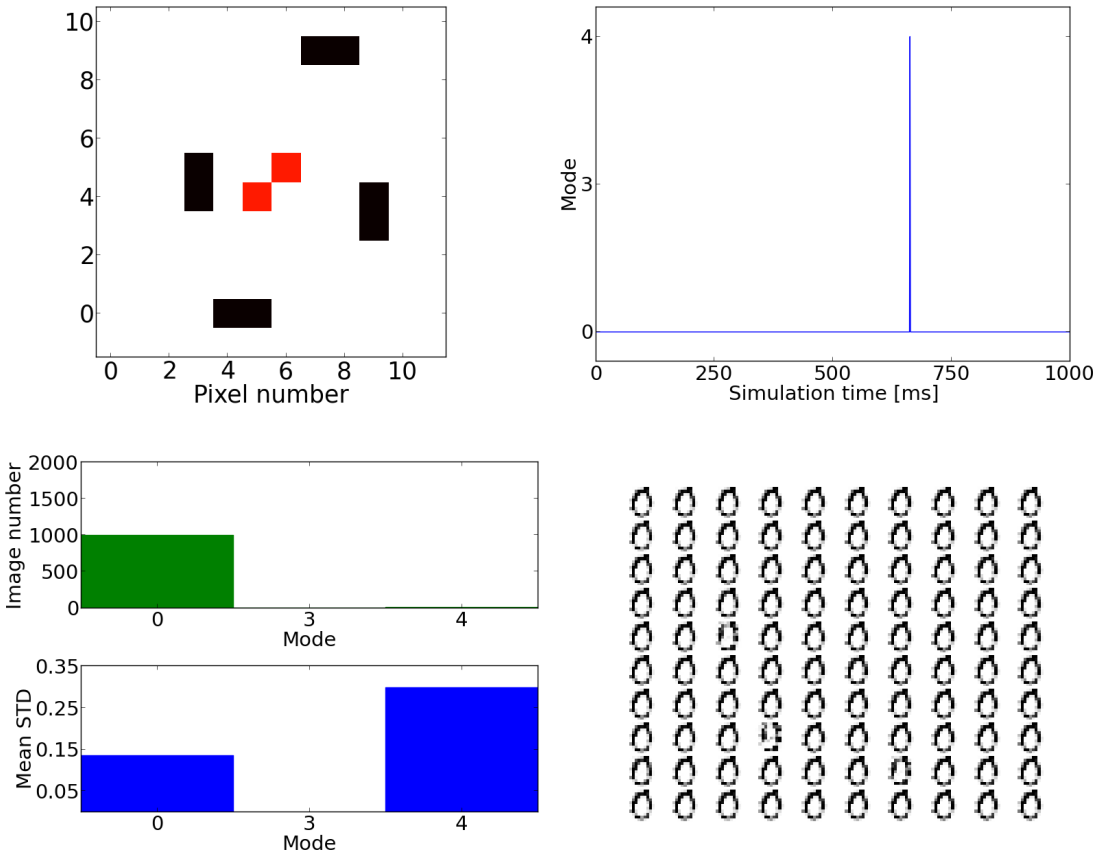


Figure 4.17.: Pattern completion in the LIF-based RBM.
Top left: Eight feature points of image 0 are clamped to 1 (black) and two feature points of image 3 and 4 are clamped to 0 (red). **Top right**: Evolution of network modes, $T_{sim} = 10^3$ ms. The network stays in mode 0 almost all the time during simulation (it only switches once, for 1 ms, to mode 4). **Bottom left**: Distribution of generated images and the mean STDs in each mode. **Bottom right**: A series of consecutive image samples (T_{sim}: 625 - 725 ms) generated by the network (by column). The network continuously generates the 'correct' image (consistent with the observed pixels).

The results show that, clamping on a few feature points, the LIF-based RBM is able to produce a series of complete corresponding images. This demonstrates the ability of the LIF-based RBM to perform pattern completion.

Ambiguous pattern recognition (pattern rivalry)

If the generative model is given ambiguous input, it will try to complete both corresponding patterns simultaneously. Since, however, the two patterns are mutually exclusive (by

the definition of 'ambiguity'), the temporal evolution of the model will exhibit so-called pattern rivalry, i.e., it will alternate between generating either of the two competing patterns. For our case, ambiguous input can be represented by clamping on some feature points shared by multiple images from the training set.

In practice, we clamped six feature points shared by images 0 and 4 to 1, and clamped 1 feature point of image 3 to 0 in order to exclude it from the output. The result of a simulation for 10^3 ms is shown in Fig. 4.18.

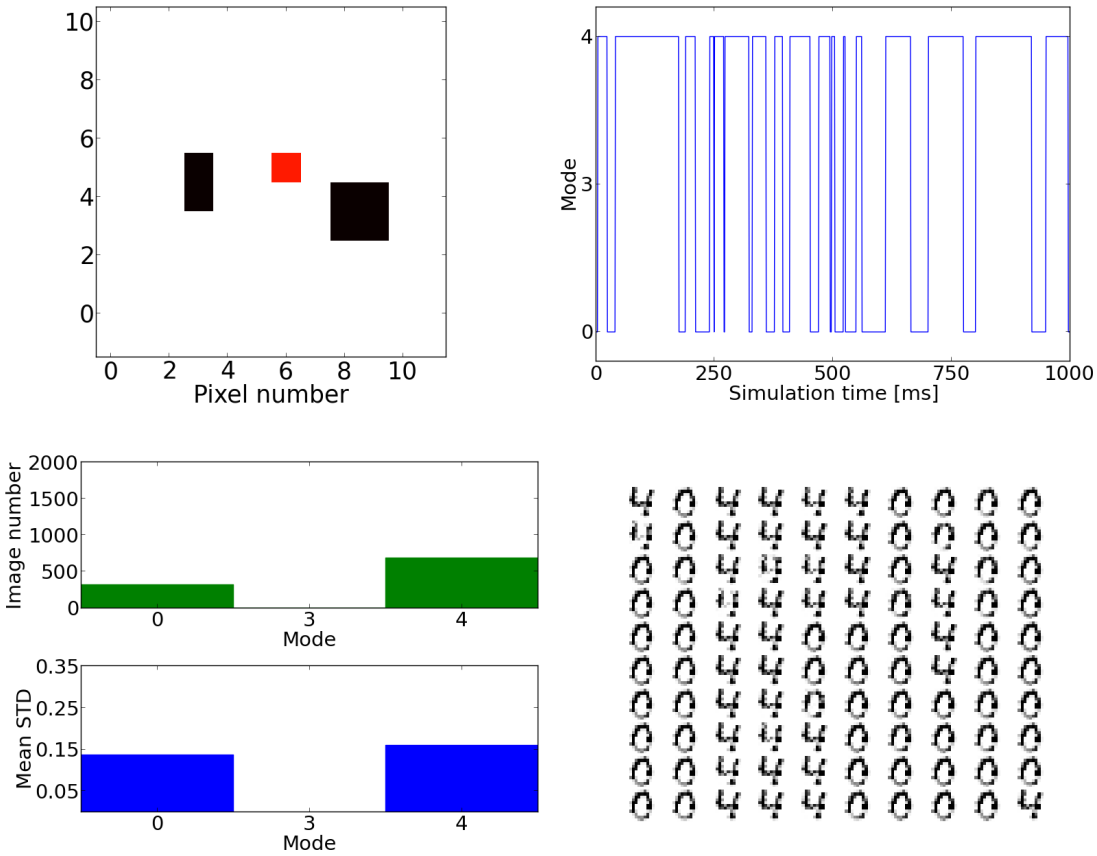


Figure 4.18.: Ambiguous pattern recognition (pattern rivalry).
Top left: Six feature points shared by image 0 and 4 are clamped to 1 (black) and 1 feature point of image 3 is clamped to 0 (red). **Top right**: Evolution of network modes, $T_{sim} = 10^3$ ms. The network switches between mode 0 and 4 frequently during simulation. **Bottom left**: Distribution of generated images and the mean STDs in each mode. The network stays mainly in mode 0 and 4 during simulation.**Bottom right**: A series of consecutive image samples (T_{sim}: 450 - 550 ms) generated by the network (by column). The network produces image 0 and 4 due to the shared clamping.

The results show that, with an ambiguous clamping setup, the output of the network switches between images sharing the clamped-to-1 feature points (images 0 and 4), and never goes to the image corresponding to the feature points being clamped to 0 (image 3). This demonstrates the ability of the LIF-based RBM to perform stochastic inference, i.e., given a certain condition (ambiguous input), it will sample from the correct posterior (i.e., alternate between the corresponding modes).

The above two experiments were intended as a preliminary test and a proof of principle. With larger network sizes, our model can be applied to recognition tasks of more complex patterns.

4.5.3. Learning 15 images of handwritten digits

To test the performance of the LIF-based RBM in learning more complex data, we further trained the model to learn 15 images of MNIST handwritten digits. The images were the same as in section 4.4.2, as shown below:

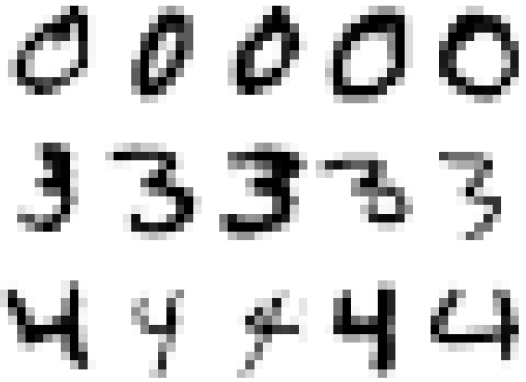


Figure 4.19.: Training images: 15 images of handwritten digits from classes '0', '3' and '4' taken from the MNIST database, reduced to 12×12 pixels.

We set the visible and hidden unit number of the LIF-based RBM to 144 and 100, respectively, which was the same as in the AMN simulation. The mixing time of the network was evaluated and chosen to be 100 ms. To save computation time, the network was initialized with the trained AMN model parameters in section 4.4.2. We further trained the network for $4.5 \cdot 10^3$ parameter updates (cost 10 days on DOPAMINE) with a constant small learning rate $\eta = 3 \cdot 10^{-5}$. After training, we ran a simulation to evaluate the network. For comparison, the simulation result of the direct parameter translation without training is shown in figure 4.20. The result after training is shown in Fig. 4.21.

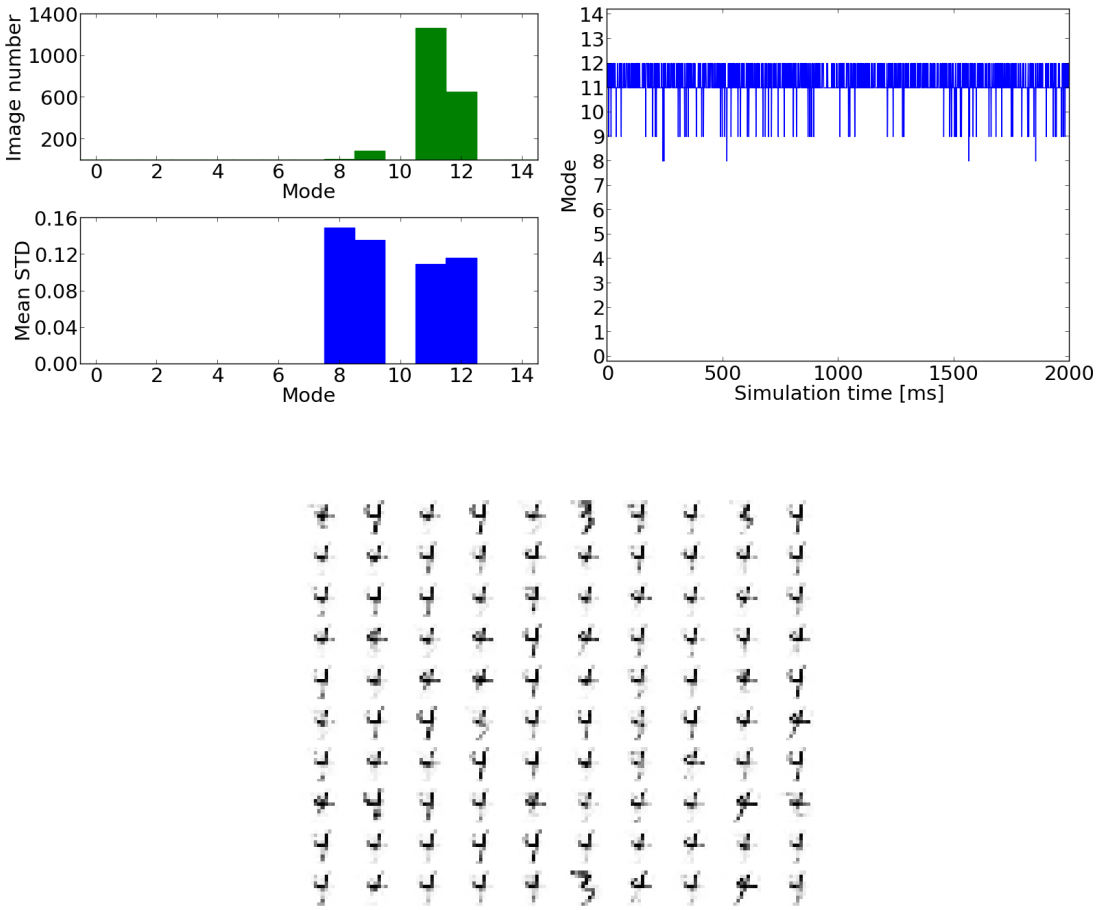


Figure 4.20.: Statistics of generated images from a simulation with $T_{sim} = 2 \cdot 10^3$ ms, after direct parameter translation without training.
Top left: Distribution of generated images and the mean STDs in each mode. Images generated by the network are only located in 4 modes (out of 15). **Top right**: Evolution of network modes. The network switches mainly between three modes (only rarely going to a fourth). **Bottom**: A series of consecutive image samples (T_{sim}: 1600 - 1700 ms) generated by the network (by column).

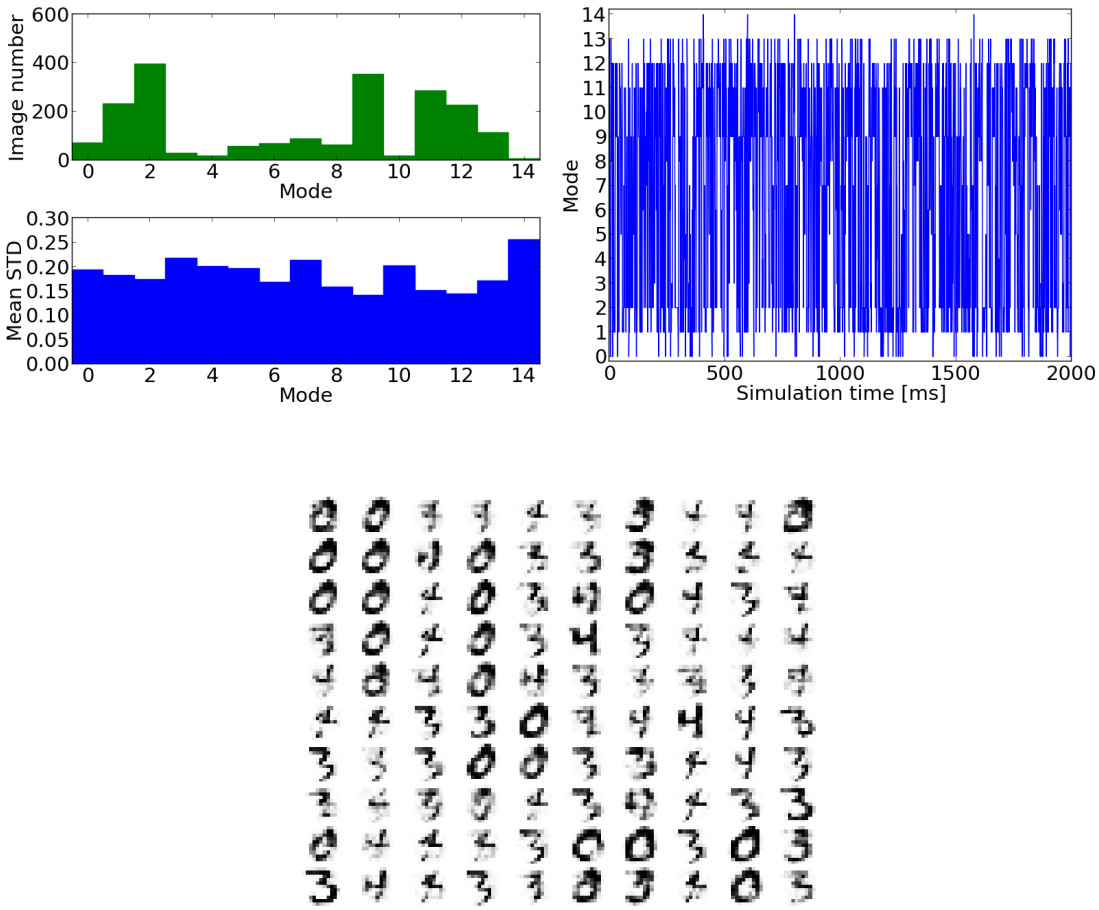


Figure 4.21.: Statistics of generated images from a simulation with $T_{sim} = 2 \cdot 10^3$ ms, after training the LIF-based RBM to learn 15 images using the CAST algorithm. **Top left**: Distribution of generated images and the mean STDs in each mode. Images generated by the network cover all modes. The mean STDs are not quite as good as the ones obtained from the AMN simulation (see Fig. 4.12), but should improve significantly with more training steps. **Top right**: Evolution of network modes. The network is able to switch between all modes with high frequency. **Bottom**: A series of consecutive image samples (T_{sim}: 450 - 550 ms) generated by the network (by column).

The results show that without training, the network only stayed in 4 modes during simulation. After training using the CAST algorithm, the network is able to produce images covering all 15 modes. The distribution of generated images is not perfectly flat, this is likely due to insufficient training, since we only trained for $4.5 \cdot 10^3$ times with small learning rates. The mean STDs range from 0.15 - 0.25, which are still not as good as the ones obtained from the AMN experiment (see Fig. 4.12), but this should also be

improved with more training. Another possibility to improve the results would be using the current-based LIF neuron model, since difficulties related to changing the recurrent conductances during training (see section 3.2.2) can be avoided. Most importantly however, computing and using average values $\langle z_i z_j \rangle_{\text{data}}$ in the training algorithm is likely to yield significant improvements for both the convergence speed and the ultimate quality of the training.

A comparison of CSL estimates during the training of three networks is shown in Fig. 4.22.

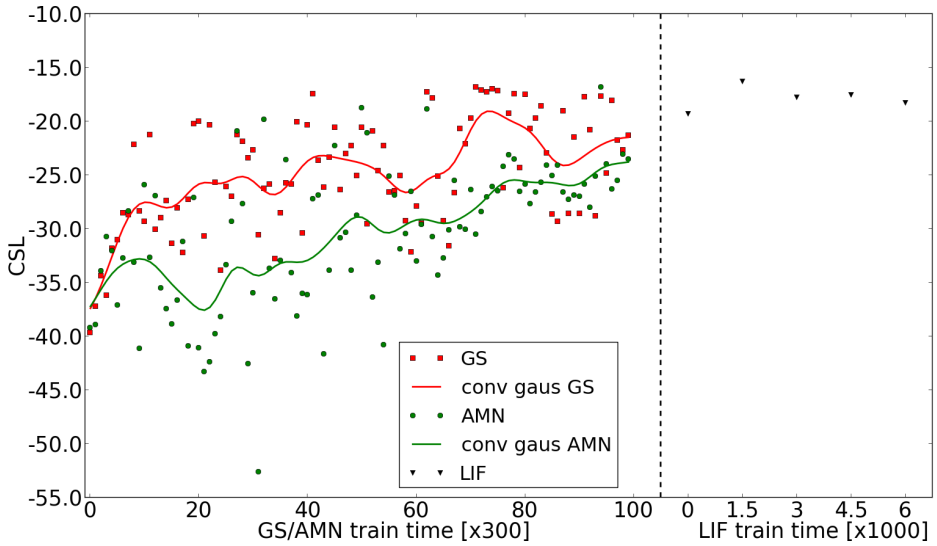


Figure 4.22.: Evolution of CSLs of the traditional GS-based RBM, the AMN-based RBM and the LIF-based RBM. Both the GS-based and AMN-based RBM were trained for $3 \cdot 10^4$ times with the same exponentially decreasing learning rates as Eq. 4.27. The LIF-based RBM was trained with constant learning rates $\eta = 3 \cdot 10^{-5}$ for $4.5 \cdot 10^3$ times, with initializing parameters translated from the trained model parameters of the AMN-based RBM.

The learning rates used in the training of the GS-based and AMN-based RBM are the same as Eq. 4.27. For both networks we collected the model parameters at 100 time points during the training. Due to time limitations, we only recorded model parameters at 5 time points for the LIF-based RBM.

Due the same reason mentioned in section 4.5.2 (using individual data points instead of averages during training), fluctuations exist in the evolution of CSLs. Compared to the training of 3 images, the model converges more slowly, as seen in the comparatively shallow CSL temporal profile. This is not unexpected, as this task is significantly more demanding. One can also see that in general the GS-based RBM performs better than the AMN-based RBM. Due to the final training steps, the LIF-based RBM shows a

significant improvement in the CSL. With more data points and longer training times, this tendency should become even more evident.

Identically to the previous section, all images were obtained by simply running the network, without any algorithms to facilitate mixing. It was interesting that the network still mixed so well between all the modes. For the AMN-based RBM, however, a simulation with plain neural sampling without the aid of the AST algorithm led to bad mixing. The simulation result of the AMN-based RBM (initialized with the same network parameters as in section 4.4.2) with plain neural sampling is shown in figure 4.23.

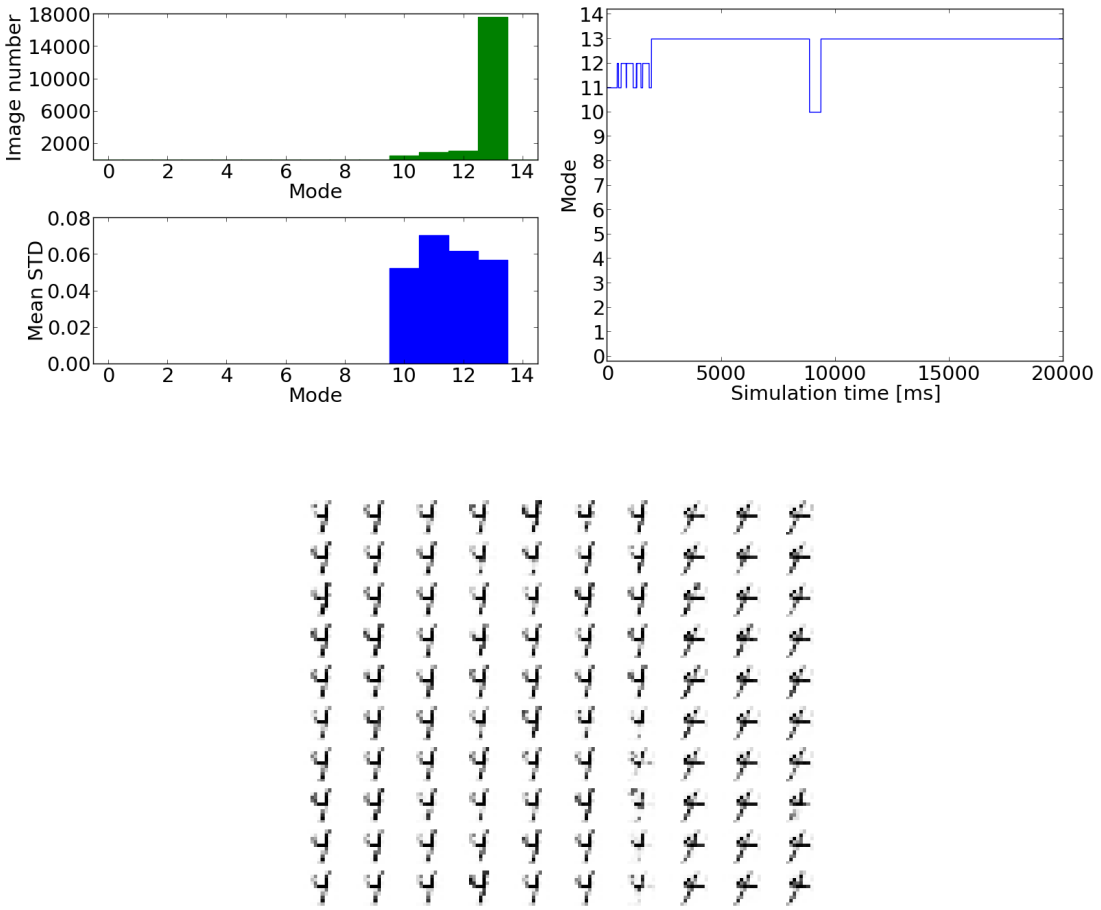


Figure 4.23.: Statistics of generated images from the AMN-based RBM for a simulation with $N_{steps} = 2 \cdot 10^4$, using plain neural sampling .
Top left: Distribution of generated images and the mean STDs in each mode. Images generated by the network are only located in 4 modes. **Top right**: Evolution of network modes. The network is not able to mix among all the modes, which is in contrast with Fig. 4.12 (use the AST algorithm) **Bottom**: A series of consecutive image samples (T_{sim}: 1200 - 1300 ms) generated by the network (by column).

The comparison shows the mixing advantage of the LIF network compared to the AMN-based network. While being extremely useful, this 'feature' of the LIF-based RBM is, at the moment, not well understood. A further investigation needs to be carried out, in particular for simulations in larger networks with more network modes.

4.6. Deep Boltzmann machines

In the previous sections, we showed the capability of RBMs to learn complex training patterns. Adding additional layers of hidden units to RBMs, one obtains so-called deep Boltzmann machines (DBMs). Previous works, such as (*Bengio and LeCun*, 2007) prove that adding layers of hidden units can theoretically improve the efficiency of representing complex distributions. Additionally, several authors, e.g., (*Salakhutdinov and Hinton*, 2009) (*Srivastava and Salakhutdinov*, 2012) have shown that DBMs can significantly outperform many other models on learning difficult datasets including MNIST handwritten digit, NORB (*LeCun et al.*), and various classification tasks.

In this section, we will briefly discuss the concept of DBMs and their corresponding learning algorithms, without going into detailed mathematical principles. Currently, we only trained a traditional DBM and will present these results in the following.

Conventionally, people refer to BMs with multiple hidden layers as deep Boltzmann machines (DBMs). A DBM can also be viewed as a (vertical) stack of several RBMs. Fig. 4.24 shows a three-layer[3] DBM.

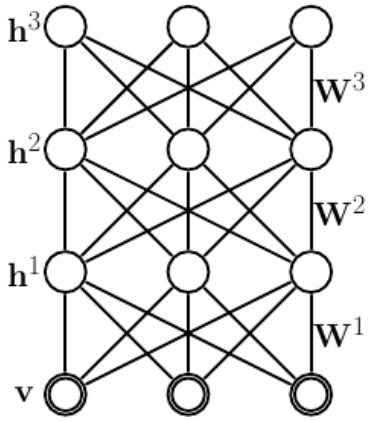


Figure 4.24.: A three-layer DBM. It can be viewed as three RBMs stacked together (Image is taken from *Salakhutdinov and Hinton*, 2009).

For a two-layer DBM, the energy of the state $\{\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2\}$ is defined as:

$$E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2) = -\sum_i a_i v_i - \sum_j b_j h_j^1 - \sum_k c_k h_k^2 - \sum_{i,j} W_{ij}^1 v_i h_j^1 - \sum_{j,k} W_{jk}^2 h_j^1 h_k^2 \,, \quad (4.28)$$

where $\mathbf{W}^1$, $\mathbf{W}^2$ are symmetric, zero-diagonal matrices that contain the visible-to-hidden and hidden-to-hidden weights. a_i, b_j and c_k are biases of the corresponding units.

[3] In (*Salakhutdinov and Hinton*, 2009), the layer number of a DBM is referred to as the number of its hidden layer, we will follow this convention here.

The probability of a visible state $\mathbf{v}$ in the DBM is defined as:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}^1, \mathbf{h}^2} \exp\left[-E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2)\right] \ . \tag{4.29}$$

The conditional distributions over the visible and the hidden units are given by logistic functions:

$$p(v_i = 1|\mathbf{h}^1) = \sigma(a_i + \sum_j W_{ij}^1 h_j^1) \ , \tag{4.30}$$

$$p(h_j^1 = 1|\mathbf{v}, \mathbf{h}^2) = \sigma(b_j + \sum_i W_{ij}^1 v_i + \sum_k W_{jk}^2 h_k^2) \ , \tag{4.31}$$

$$p(h_k^2 = 1|\mathbf{h}^1) = \sigma(c_k + \sum_j W_{jk}^2 h_j^1) \ . \tag{4.32}$$

4.6.1. Learning algorithm

Salakhutdinov and Hinton (2009) introduced an efficient layer-by-layer pretraining algorithm for the DBM, including a series of specific techniques. Additionally, a mechanism of discriminative fine-tuning was proposed, which can further improve the model after the training.

Currently, we only implemented the concept of layer-by-layer pretraining, with some modifications from the original method. As this will constitute the main focus of future research, we only describe it intuitively here, without going into detailed mathematical principles.

In the layer-by-layer pretraining, the RBM in the first layer is trained like a single RBM, and then serves as a feature detector for the RBM in the second layer. More specifically, after training, the first-layer RBM then performs a non-linear transformation on the input data and produces, as an output, certain states of its hidden units. These are used as input for the second-layer RBM. The second-layer RBM is trained using the hidden state of the first-layer RBM as its visible state. For the training of the RBM in each layer, we applied the CAST algorithm instead of the CD or PCD used in the original work (*Salakhutdinov and Hinton*, 2009).

4.6.2. Result

We trained the model with 100 MNIST handwritten digits, similarly to section 4.3, as shown below:

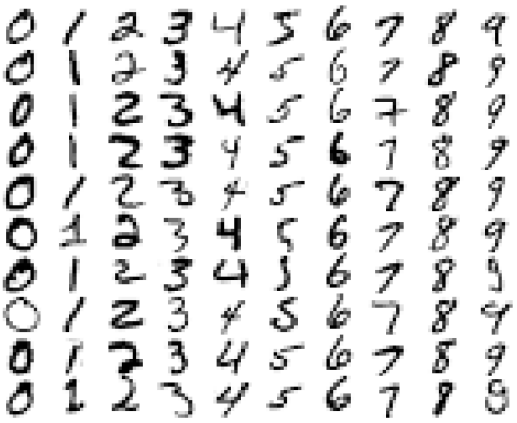


Figure 4.25.: Training data: 100 images of handwritten digits taken from the MNIST database and reduced to 12×12 pixels.

The unit number of the visible layer, hidden-layer-1 and hidden-layer-2 were set to 144, 90 and 180, respectively. For the choice of the number of hidden units in each layer, we used a similar setup as *Salakhutdinov and Hinton* (2009), with a larger number of hidden units in layer $\mathbf{h}^1$ compared to layer $\mathbf{h}^2$. However, a decreasing number of hidden units per layer can also be an option, as in (*Hinton and Salakhutdinov*, 2006). The model was trained for $2 \cdot 10^5$ parameter updates, with 10^5 for each RBM. Both RBMs used the same learning rates:

$$\text{for } n = 0 - 7 \cdot 10^4,\ \eta(n) = (10^{-4} - 10^{-2} + 1)^{\frac{n}{7 \cdot 10^4}} + 10^{-2} - 1\ ,$$

$$\text{for } n = 7 \cdot 10^4 - 10^5,\ \eta(n) = (3 \cdot 10^{-5} - 10^{-4} + 1)^{\frac{n - 7 \cdot 10^4}{3 \cdot 10^4}} + 10^{-4} - 1\ .$$

After training, the result of a simulation for $\mathrm{N_{steps}} = 8 \cdot 10^4$ can be seen in Fig. 4.26.

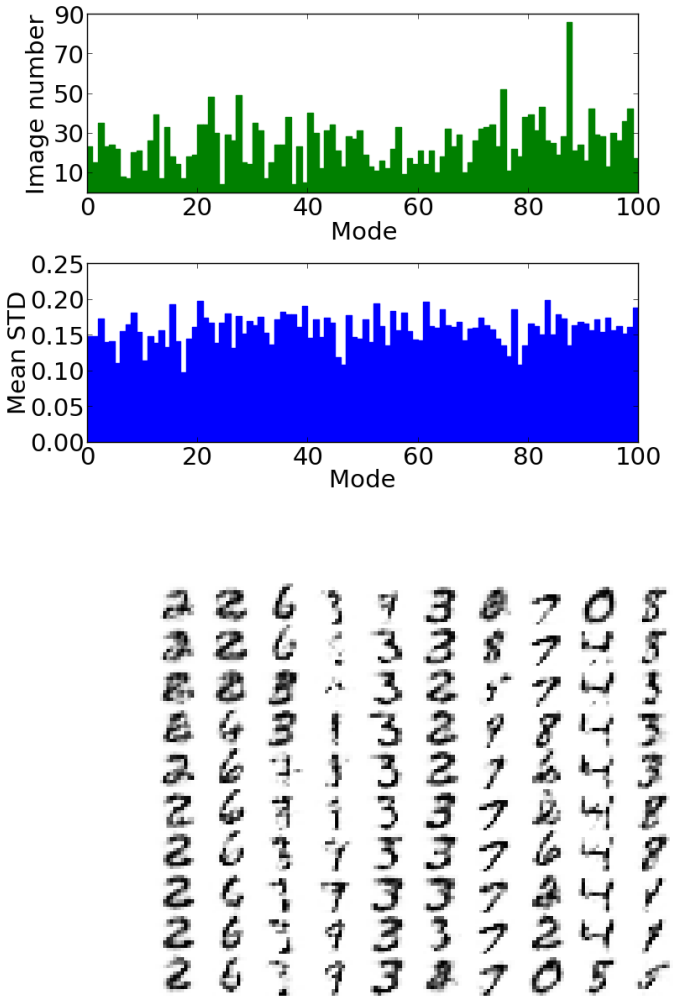


Figure 4.26.: Statistics of generated images from a simulation of $N_{steps} = 8 \cdot 10^4$.
Top: Distribution of generated images and the mean STDs in each mode. Images generated by the network cover all modes. **Bottom**: A series of consecutive image samples generated by the network (by column). The network switches between different image classes frequently.

For the generation of images, we first ran a simulation on the second-layer RBM using AST and obtained a series of states $\{\mathbf{h}^1\}$. These states were then used for the initialization of the first-layer RBM. Based on each state $\mathbf{h}^1$, the firing probabilities of the visible units were calculated and taken as the pixel values of the generated image.

For the learning of 100 images, the result looks nice even without further implementation of other techniques. For larger and more complex training data, the use of additional mechanisms such as discriminative fine-tuning (*Salakhutdinov and Hinton*,

2009) is expected to allow significant improvements. We have already commenced investigations of DBMs based on AMNs and LIF neurons and will continue pursuing this line of research in the future.

4.7. Summary

In this chapter we explored the implementation of RBMs with AMNs and LIF neurons. A series of corresponding learning algorithms were developed and the training results proved their feasibility. The capability of the LIF-based RBMs to learn unlabeled data was demonstrated. Additionally, we showed the potential of the LIF-based BM to perform pattern recognition tasks. This work paves the way for the exploration of LIF-based deep Boltzmann machines, where the LIF-based RBMs serve as building blocks.

The LIF-based RBMs show great potential in learning and generating datasets with high variability. For learning 15 images, the total neuron number required is less than twice the pixel number of the image, and we expect to be able to decrease it even further without significant impairments in performance. Future improvements of the code used for instantiating and updating the LIF-based BMs (some of them already implemented but not yet tested) will allow us to expand the size of the LIF-based RBM and apply it to learn the full MNIST dataset, other visual datasets (e.g., NORB), or even speech recognition tasks, where RBMs also showed high performance (*Jaitly and Hinton*, 2011).

Finally, a successful implementation of the LIF-based RBM on the neuromorphic hardware will greatly profit from its fast simulation speed, especially when the learning can be performed on-line (a neuromorphic implementation of an STDP-based EM[4] learning rule, inspired by *Nessler et al.* (2009), is currently being investigated by Oliver Breitwieser).

[4]expectation maximization

5. Hardware implementation of LIF-based BMs

This chapter will give a short introduction to the neuromorphic hardware developed within our group. We focus, in particular on the wafer-scale device and its constituent HICANN chips. Following the hardware description, we present our progress on the hardware implementation of LIF-based BMs and describe several important problems found during the process.

5.1. Introduction of the neuromorphic hardware

Compared to the traditional numerical approach to the modeling of biological neural networks, so-called neuromorphic hardware directly implements neuron and synapse models with circuits on a silicon substrate using VLSI[1] technology. This approach offers several advantages over traditional simulations on von Neumann architectures: it is inherently (massively) parallel and, in general, requires less power (or energy per synaptic event). Additionally, in the particular case of the BrainScalesS physical model device, the network components' dynamics inherently evolve at a speedup of 10^4 compared to their biological archetypes.

Currently, within the BrainScaleS[2] project (and its predecessor FACETS[3]), a wafer-scale system (see Fig. 5.1) was designed and built. Each wafer contains 352 HICANN[4] chips with high connection density between individual modules (*Schemmel et al.*, 2010).

[1] Very Large Scale Integration
[2] Brain-inspired multiscale computation in neuromorphic hybrid systems
[3] Fast Analog Computing with Emergent Transient States
[4] High Input Count Analog Neural Network

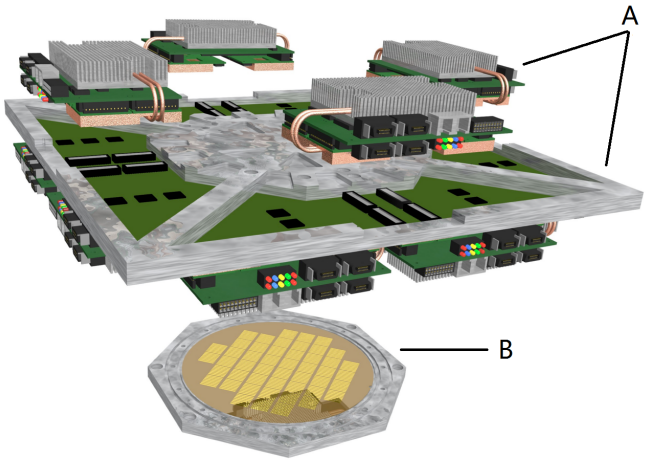


Figure 5.1.: Design drawing of the wafer-scale neuromorphic device. A: Structures responsible for power supply and communication with other devices. B: Wafer with HICANN chips.

The HICANN chip, as shown in Fig. 5.2, consists of two blocks with 256 AdEx[5] neurons and 65536 synapses each. The AdEx neuron model can be reduced to the LIF neuron model (as required by our theory) by deactivating certain parameters.

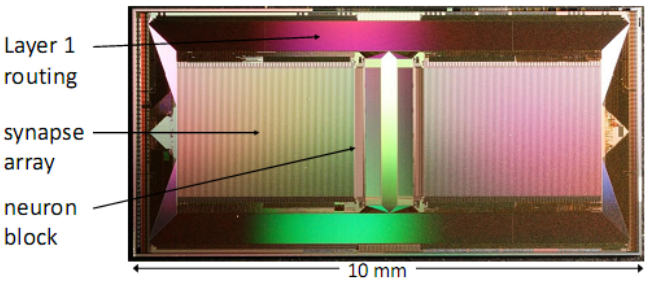


Figure 5.2.: Photograph of the HICANN (Image is taken from *Schemmel et al.*, 2010).

Depending on the setup of certain parameters, the temporal evolution of the neural components is expected to be 10^3 to 10^5 times faster compared with biological real-time.

[5] Adaptive Exponential Integrate and Fire Model (*Brette and Gerstner*, 2005)

5.2. Implementation of LIF-based BMs in the neuromorphic hardware

Our[6] first goal was to implement a 3-neuron BM with random weights and biases on the HICANN chip. As a first test, we created a single excitatory Poisson noise source and injected its output into one neuron to check its spiking patterns. We observed regular membrane potential responds, a segment of which is shown in Fig. 5.3. The observed time course of the membrane potential (Fig. 5.3) was somewhat as expected, although the lack of calibration hindered a precise tuning of the neuron and synaptic parameters.

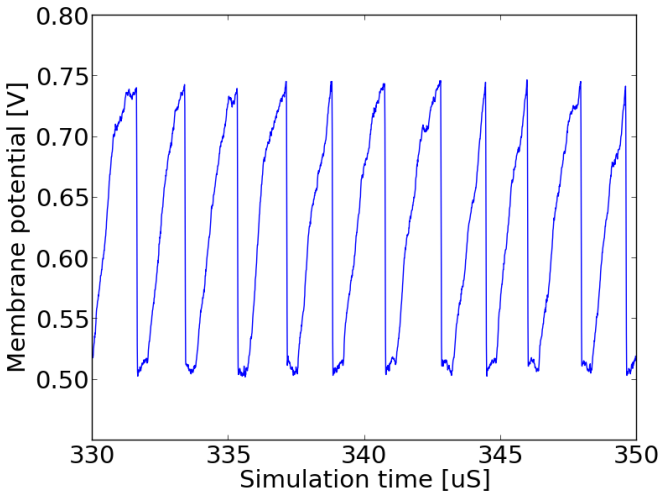


Figure 5.3.: Membrane potential profile under an excitatory Poisson noise source.

Following the setup used in the previous chapters (excitatory & inhibitory Poisson noise for each neuron), we then attempted to add another (inhibitory) Poisson noise source to the neuron. However, we found that the neuron was only able to receive noise sources with address 0 (HICANN.DriverDecoder(0)). This bug made the setup with two Poisson noise sources for each neuron impossible at that time. More recently, Dimitri Probst found a workaround for this bug with a more complicated setup.

Since the bug seemed to not occur for regular spike trains (generated by the same sources as the Poisson noise), we replaced the noise stimulus with regular stimulus in order to at least investigate the effects of simultaneous excitation and inhibition. Each neuron was injected with regular excitatory and inhibitory spike trains with a constant frequency of 400 Hz. However, some unexpected behavior occurred here as well. We found that, for certain neurons, a long-term noise injection would cause something like a bistable state (see Fig. 5.4.). The neuron would witch between a non-spiking mode with oscillatory membrane potential (as expected due to the regular nature of the stimulus)

[6]This work was performed in close collaboration with Dimitri Probst

and a fast-spiking mode - as if, for a short period of time, the threshold would be set to a lower value. The asymmetry between excitatory and inhibitory was a consequence of the lack of calibration for the synaptic input.

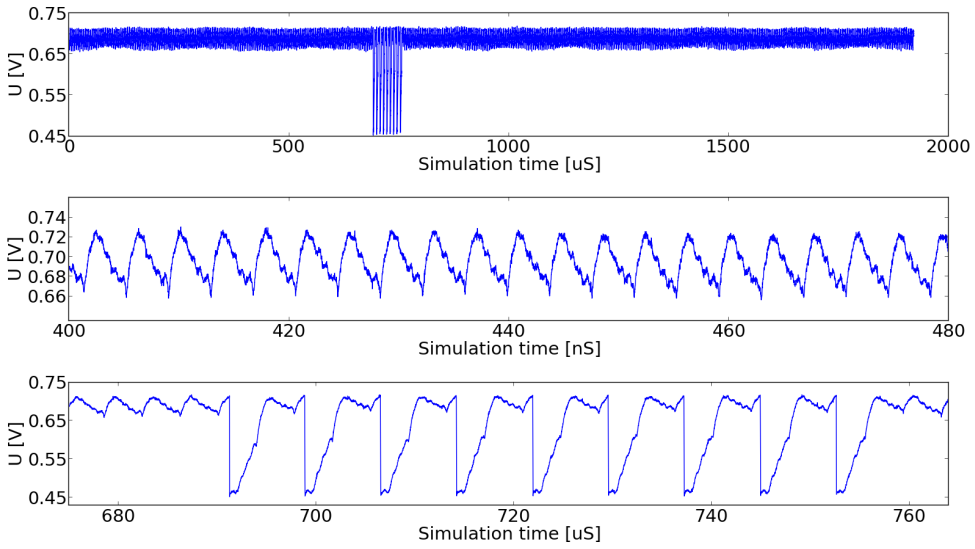


Figure 5.4.: **Top**: A complete record of the membrane potential. **Middle**: Oscillatory membrane potential in the non-spiking mode. **Bottom**: Membrane potential in the fast-spiking mode.

In some runs, we found that the bistable behavior can be eliminated for certain parameter settings. In any case, more trials and a more thorough investigation are needed for a full understanding of this phenomenon.

In conclusion, during the two weeks of preliminary exploration of the implementation of a simple LIF-based BM on HICANN, we found a series of bugs and problems. Due to the time limitations, only a few of them were investigated and discussed to a very limited degree. Some problems can be temporarily avoided by special setups or by picking a particular range of certain parameters. However, for a concrete implementation of a 3-neuron BM, more work is needed for a full investigation of all the problems.

One challenge will be the separate investigation and remedy of certain bugs which occur simultaneously on the hardware. We found it particularly difficult to judge whether a 'new' problem was caused by previously observed bugs or new ones.

Consequently, we consider a thorough investigation of the hardware neuron and synapse dynamics - and, in particular, of the way they are influenced by parameter settings - indispensable and of highest priority before the implementation of BMs can be further pursued.

6. Summary & Outlook

6.1. Summary

Throughout this thesis, we explored functional applications of LIF networks to standard machine learning tasks. Learning algorithms were developed for LIF-based multilayer Boltzmann machines and their efficiency was proved for several training tasks.

In chapter 2, we have discussed various implementations of Boltzmann machines, including some basic considerations about learning in these networks. Starting from the original Boltzmann machine design based on abstract binary units, we discussed how this concept can be gradually refined towards a more biological implementation. Following the neural sampling theory by Buesing et al., we have shown how abstract model neurons (AMNs) can replace the binary variables from the original design. Then, based on the LIF sampling theory by Petrovici et al., we have shown how networks of LIF neurons can achieve the correct dynamics to sample from Boltzmann distributions, thereby establishing an equivalence between classical Boltzmann machines and LIF networks. Starting from basic maximum-likelihood optimization, we have also discussed how the so-called contrastive divergence algorithm by Hinton et al. can efficiently approximate maximum likelihood learning. Based on the theory developed in chapter 2, we were able to use networks of LIF neurons to implement increasingly complex Boltzmann machines.

In chapter 3, we constructed Boltzmann machines consisting of several hundred units, thereby evaluating the performance of large networks of LIF samplers. In particular, we used these networks to model the well-known phenomenon of perceptual multistability. Starting from a trained network of theoretically optimal AMNs, we have shown how the resulting network parameters can either be translated directly to an LIF network or how the latter can be trained independently for the task at hand. For training the LIF network, we adapted the contrastive divergence algorithm accordingly. In both cases, we were able to demonstrate a good performance of the LIF network, in particular for the case where it was trained independently. We have provided a detailed discussion of these results, emphasising the effects of synaptic conductances on the translation process. Additionally, in order to verify the applicability to analog neuromorphic circuitry, we have repeated the experiment with noised neuron parameters. Using the "self-calibrating" framework designed by Mihai Petrovici, we have demonstrated no loss in performance due most fixed-pattern noise effects, thereby encouraging a future implementation on the neuromorphic hardware.

In chapter 4, we extended our scope by discussing deep learning architectures and

their implementations with LIF networks. Due to the more complex, multilayered structure, as well as the significant differences in training (training data is usually only available for the so-called "visible" layer of a deep architecture), more complex and efficient learning algorithms were necessary. A series of learning algorithms were compared and combined to form the so-called CAST algorithm, which was shown to perform well in learning patterns with high variability. We then modified this algorithm to make it applicable to networks of both AMN and LIF neurons. Based on this modified algorithm, we were able to demonstrate the ability of these networks to learn generative models of complex input patterns. As training data, we have used samples of varying size from the MNIST handwritten digit database, which by now has become a standard in machine learning. Furthermore – and somewhat serendipitously – we found the LIF networks to inherently exhibit better mixing properties than their more abstract counterparts.

By combining several of the previously discussed restricted Boltzmann machines, one obtains so-called deep Boltzmann machines, which have shown state-of-art performance in various machine learning tasks. In the last section of chapter 4, we briefly described our current progress on these architectures, together with a preliminary training result of a traditional three-layer deep Boltzmann machine.

Finally, first steps towards the hardware implementation of LIF-based Boltzmann machines were presented in chapter 5. At the current stage of development, we encountered a series of bugs related to the on-chip Poisson sources, which are essential for implementing the stochasticity of individual LIF sampling units. Before attempting any further steps, these problems need to be better understood and fixed appropriately.

6.2. Outlook

The immediate next step will be the implementation of deep Boltzmann machines with LIF neurons. Also, with improvements in the software implementation, the training should become faster and better. Additionally, in future work, the current algorithms and evaluation methods will be improved by including mechanisms such as discriminative fine-tuning and annealed importance sampling (AIS) (*Salakhutdinov and Hinton*, 2009).

An interesting theoretical problem will be the translation of the learning algorithms to biological dynamics. *Nessler et al.* (2009) have shown, for example, how STDP can be used to model on-line expectation maximization learning. Additionally, the changes in temperature required by the simulated tempering algorithm should find an equivalent in the noise level of the network, which can be modulated, for example, by inhomogeneous Poisson sources[1].

The developed learning framework for LIF-based Boltzmann machines show a vast application prospect, considering the excellent performance of classical, multilayer

[1] Personal communication with Mihai Petrovici

Boltzmann machines demonstrated in recent literature (*Salakhutdinov and Hinton*, 2009), (*Srivastava and Salakhutdinov*, 2012). We plan to extend the size of our current LIF-based Boltzmann machines (both the number of layers and the number of units per layer) and apply them to more complex learning tasks, including, for example, the full MNIST or NORB datasets or non-visual learning tasks such as speech recognition. The discovered mixing properties of the LIF network might be highly relevant for these tasks and therefore need to be studied in detail.

Additionally, the developed LIF-based learning framework can be further refined and integrated with other stochastic models, such as the LIF-based Bayesian networks studied by Dimitri Probst.

Last but not the least, we are looking forward to a successful implementation of LIF-based Boltzmann machines on the accelerted BrainScaleS neuromorphic hardware. The huge (10^4) acceleration factor of the hardware, combined with robust on-line learning methods, will likely allow "neuromorphic Boltzmann machines" to rival state-of the art software implementations.

Bibliography

Bengio, Y., and Y. LeCun, Scaling learning algorithms towards ai, *Large-Scale Kernel Machines*, *34*, 1–41, 2007.

Bengio, Y., and L. Yao, Bounding the test log-likelihood of generative models, *arXiv preprint arXiv:1311.6184*, 2013.

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.

Brunel, N., and S. Sergi, Firing frequency of leaky integrate-and-fire neurons with synaptic current dynamics, *Journal of Theoretical Biology*, *195*, 87–95, 1998.

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.

Fiser, J., P. Berkes, G. Orbán, and M. Lengyel, Statistically optimal perception and learning: from behavior to neural representations, *Trends in cognitive sciences*, *14*(3), 119–130, 2010.

Hinton, G., A practical guide to training restricted boltzmann machines, *Momentum*, *9*(1), 2010.

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

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

Hinton, G. E., S. Osindero, and Y.-W. Teh, A fast learning algorithm for deep belief nets, *Neural computation*, *18*(7), 1527–1554, 2006.

Jaitly, N., and G. Hinton, Learning a better representation of speech soundwaves using restricted boltzmann machines, in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 5884–5887, IEEE, 2011.

Knill, D. C., and A. Pouget, The bayesian brain: the role of uncertainty in neural coding and computation, *TRENDS in Neurosciences*, *27*(12), 712–719, 2004.

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

LeCun, Y., F. J. Huang, and L. Bottou, Learning methods for generic object recognition with invariance to pose and lighting, in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2, pp. II–97, IEEE.

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

Nessler, B., M. Pfeiffer, and W. Maass, Stdp enables spiking neurons to detect hidden causes of their inputs., in *NIPS*, pp. 1357–1365, 2009.

Petrovici, M., Function vs. substrate: Theory and models for neuromorphic hardware, Ph.D. thesis, 2014.

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

Ricciardi, L. M., and L. Sacerdote, The ornstein-uhlenbeck process as a model for neuronal activity, *Biological Cybernetics*, *35*, 1–9, 1979.

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.

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.

Schwartz, M.-O., Reproducing biologically realistic regimes on a highly-accelerated neuromorphic hardware system, Ph.D. thesis, Universität Heidelberg, 2013.

Srivastava, N., and R. Salakhutdinov, Multimodal learning with deep boltzmann machines., in *NIPS*, pp. 2231–2239, 2012.

Sutskever, I., and T. Tieleman, On the convergence properties of contrastive divergence, in *International Conference on Artificial Intelligence and Statistics*, pp. 789–795, 2010.

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.

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.

A. Acknowledgments

For the “Lebendigen Geist” of all the living creatures of Im Neuenheimer Feld.

I would like to express my gratitude to all those who helped me during the writing of this thesis.

I would like to express my appreciation and gratitude to Prof. Dr. Karlheinz Meier and Dr. Johannes Schemmel for supporting my thesis.

I am deeply grateful for the (huge amount of) helps and advises from Mihai Petrovici, without whom this thesis won't be how it looks like now.

Special thanks should go to Ilja Bytschok who have made (also a huge amount of) nice modifications on the English expressions of this thesis.

My deepest gratitude goes to my parents for their loving considerations and spirit support for all these years. I also owe my sincere gratitude to my friends for their accompany in Heidelberg, where we shared a lot of happiness.

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, January 21, 2015

.....
(signature)