Master's Thesis

Learning temporal relationships between hidden causes in networks of spiking neurons

Institute for Theoretical Computer Science
Graz University of Technology

Submitted by:
David Kappel

Supervisor:
O.Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Wolfgang Maass

January 30, 2011

Masterarbeit

Lernen zeitlicher Zusammenhänge zwischen versteckten Zuständen in spikenden neuronalen Netzwerken

Institut für Grundlagen der Informationsverarbeitung
Technische Universität Graz

Vorgelegt von:
David Kappel

Betreuer:
O.Univ.-Prof. Dipl.-Ing. Dr.rer.nat. Wolfgang Maass

30. Januar 2011

Abstract

Spike time dependent plasticity (STDP) is considered to be the major mechanism for learning and adaptive behaviour in the brain. In a recent work it was shown that a purely local synaptic learn rule enables a network of spiking neurons, connected to winner-take-all circuits, to discover hidden causes. These findings suggest, that neural networks in the neocortex and the hypocampus are able to compute Bayesian inference. However, temporal correlations of the input were neglected by assuming consecutive hidden causes to be independent. Natural signals, like speech, show high correlations between nearby time windows and it is likely, that evolution has found some way to exploit these correlation in biological neural networks. In this thesis we extend the basic results on winner-take-all circuits to enable them to detect temporal relationships between hidden causes from an input spike stream. We will show that this goal can be achieved by using relatively simple extensions of the basic architecture. We will investigate the learning dynamics of such networks, and compare the results to recent data from neuroscience, and to standard machine learning paradigms such as the hidden Markov model. We study different network architectures and analyse them in terms of their computational power and biological plausibility.

Kurzfassung

Es wird angenommen, dass Spike-Zeit abhängige Plastizität (STDP) der neuronale Mechanismus für Lernen und adaptive Prozesse im Gehirn ist. In einer kürzlich veröffentlichten Arbeit wurde gezeigt, dass Schaltkreise von Neuronen, die sich innerhalb des Netzwerkes gegenseitig inhibieren, durch STDP in der Lage sind, versteckte Zustände aufzudecken. Aus diesem Ergebnis folgt, dass neuronale Netzwerke im Neocortex und dem Hippocampus, die Fähigkeit besitzen Bayes'sche Inferenz zu berechnen. Allerdings wurde hier angenommen, dass zeitlich aufeinander folgende Zustände unabhängig voneinander sind. Zeitliche Korrelationen in den Spike-Mustern wurden nicht beachtet. Natürliche Signale, wie etwa Sprachsignale, weisen starke Korrelationen zwischen nahegelegenen Zeitfenstern auf und es ist anzunehmen, dass die Evolution einen Weg gefunden hat diese im Gehirn zu verarbeiten. In dieser Arbeit wird die ursprüngliche Theorie erweitert, sodass das Netzwerk zeitliche Abhängigkeiten zwischen versteckten Zuständen entdecken kann. Es wird sich zeigen, dass dieses Ziel durch relativ einfache Erweiterungen erreicht werden kann. Wir werden die Lerndynamik solcher Netzwerke studieren und mit Ergebnissen aus den Neurowissenschaften und maschinellem Lernen vergleichen.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

date	signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen / Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am	Unterschrift

Danksagung

Mein besonderer Dank gilt...

Wolfgang Maass, für die Unterstützung dieser Arbeit und die Möglichkeit an aktueller Forschung teilzunehmen.

Bernhard Nessler, für die vielen Stunden die er dieser Arbeit geopfert hat und die kreativen Ideen die daraus entstanden sind.

Stefan Habenschuss, Markus Murschitz und Nikolaus Hammler für die anregenden Diskussionen.

meinen Eltern, Eva und Konrad die mir dieses Studium ermöglicht und stets meine Interessen und Talente gefördert haben.

meinen Großeltern, für ihre Unterstützung während meiner Studienzeit.

meinen Geschwistern, Lisa und Paul für viele unterhaltsame Stunden und aufmunternde Worte.

Susanne, für die schöne Zeit, die wir zusammen verbringen.

all meinen Freunden, Kollegen und Verwandten, die mich aufmuntern und unterstützen und ohne die diese Arbeit nicht möglich gewesen wäre.

Graz, im Jänner 2011 David Kappel

diese Arbeit ist in
englischer Sprache verfasst

Contents

List of Figures

List of Tables

1 Introduction

In the brain information between neurons is propagated by sending out uniformly shaped electric pulses, called action potentials or spikes. Virtually all input arrives at the brain in temporal patterns of such spike trains, propagated along nerval fibres. Even if we look at a static image, the saccadic movements of our eyes chop the image into dynamic ‘movies’. The same is true for the motor output of the brain. Performing tasks like juggling or playing the piano require to coordinate our arms and fingers with very precise timings. Not least speech acquisition and production require to process highly complex sequential patterns and very accurate motor control timings. Even when performing tasks like walking, the brain needs to process an endless stream of information coming form eyes, touch and equilibrium sense. Due to these facts the neural mechanisms, that enable the brain to solve these tasks - we find so difficult in machine learning - with such ease, have been of great interest by neuroscientists over the last decades.

In machine learning we speak about sequential data when data samples arrive in an ordered sequence. Such data samples often arise through measurements of time series and usually show large correlations between nearby time points [Bishop, 2006]. In classical machine learning sequential data used to play a secondary role. Data samples were usually assumed to be independent and identically distributed breaking their temporal correlation. This assumption was taken in many cases due to computational simplifications. Only in recent years processing of sequential data has attracted more and more attention. This change in paradigms was due to the increasing computational power of modern computer hardware and to the wide applicability in tasks, were the independence assumption was found to be insufficient.

One of the first statistical models that was introduced to solve this problem, was the hidden Markov model, introduced by Baum and colleges in the 1960s [Baum and Petrie, 1966]. The major assumption of this model is, that any stream of sequential data we observe in nature, can be perceived as being generated by an unobserved probabilistic state machine. If that internal state of the model was known, the independence assumption on the observed data would hold. Efficient algorithms have been derived to estimate the parameters of the model and the hidden state variables. Due to its computational simplicity the hidden Markov model has become a standard model in machine learning with applications ranging from speech processing to gene sequencing [Rabiner, 1989, Churchill, 1989].

One of the most astounding predicates of neurons is their adaptive behaviour. Many neural circuits of the brain stay plastic over their whole lifetime, being adaptive to new environments and situations. The mechanisms of this plasticity is under extensive research, but today not completely understood. Spike time dependent plasticity (STDP) is considered to be a key feature, causing adaptive behaviour. In a recent theoretical study [Nessler et al., 2010] revealed that STDP enables networks of spiking neurons to solve maximum likelihood estimation on incomplete data. This neural learning paradigm was called spiking expectation maximisation (SEM). SEM Networks showed good performance in unsupervised learning tasks and the necessary calculations could be explained on a biological background. But, in the proposed theory, the hidden causes were again assumed to be independent over time.

In this thesis we will close this gap. We will discuss models from computational neuroscience and standard machine learning approaches on learning sequential data, compare them and point out their similarities and differences. We will investigate two approaches, that introduce temporal dependences between hidden causes to the SEM network. These approaches emerge from very simple extensions to the original architecture. The first one emerges from extending the network inputs with time-lags, the second from recurrent connections that feed back the network outputs from previous time steps. We will see that the latter architecture implements a hidden Markov model. Careful analysis of both models will show that exact parameter learning can not be done using STDP learning alone. We will derive an extended learn rule that uses synaptic tagging to solve this problem. The algorithm for the hidden Markov model case was found in a collaborate work with Bernhard Nessler. We will evaluate this algorithm using numerical simulations and show that it can achieve similar performance to standard machine learning approaches.

Neurophysiological studies have shown that sequence processing tasks are accompanied by highly complex neural circuits and coding. Specialised cells in the brain encode information on a surprisingly high level of abstraction, when they are exposed to sequential data [Barone and Joseph, 1989, Averbeck et al., 2002]. On the other hand behavioural studies on humans suggest, that in the early phases of sequential data learning, only a rudimentary representation of local statistics is developed [Cleeremans et al., 1998]. It remains unclear whether the more abstract information coding is learned on top of these early statistical representations, or if the two phenomenons are independent. We will show that both phenomenons can emerge from the same neural substrate using the models introduced in this thesis. These findings suggest, that abstract neural codes can emerge from early statistical representations through a process of model refinement.

The rest of this thesis is organised as follows: In chapter 2 we will give a brief overview of the brain's sequential data processing. In chapter 3 we will review the hidden Markov model and other standard machine learning approaches. In chapter 4 we will discuss computational neuron models, compare them to the probabilistic models and introduce the SEM network and its extension. We will also talk about neural codes and extensions to STDP learning. In chapter 5 we will introduce the extended SEM approaches and discuss their advantages and disadvantages. In chapter 6 we will show experimentally that some aspects of the neural codes found in biological systems can be explained by the introduced models. In chapter 7 we conclude and give a outlook to future work.

2 Biological mechanisms for sequential data processing

The neural processing of sequential stimuli has been of great interest by the neuroscience community. Many studies have approached this problem from different research directions. In this chapter we will review some of the mechanisms that were found.

2.1 Neural codes for sequential data

The neural code, the representation of sequential data in the brain is based on, was under extensive research over the last decades. Of particular interest were experiments concerning motor planning and sequence memorisation. There are numerous studies of neural recordings from monkeys performing these tasks. A region in the brain that is highly involved in sequence processing is the prefrontal cortex. Neurons located in that area show highly complex activity patterns that exploit sequential data on a surprisingly high level of abstraction.

One early study of sequence learning and motor planning was done by [Barone and Joseph, 1989]. They recorded 302 neurons from the prefrontal cortex of macaque monkeys, which had to memorise and reproduce a sequential pattern. The monkeys were trained to push buttons on a panel in a specific order they had observed before. The panel consisted of a central fixation point and three target points, two lateral and one above the fixation point. In the first phase of the experiment the monkey had to purely observe the appearance of the target points while focusing the fixation point. The target points appeared in random order. In the second phase it had to reproduce the sequence by pressing the targets in the observed order. If the monkey correctly reproduced the sequence it was rewarded with juice. It was shown that in the first phase one set of neurons storing positions and time-relationships became active, while during execution of the sequence, another set of neurons became active that encoded the current state of the sequence.

[Barone and Joseph, 1989] identified several classes of task related neurons: One class of cells that became active in phase one, after onset of one of the targets, was denoted *visual tonic cells*. These cells were target dependent and also selective to the rank order in which the target appeared. Different groups of these cells were identified each of which encoding one of the targets activated at a specific sequential order. The activity was independent of the subsequent illumination of the other targets. It was also reported that the activity of these cells was correlated with the animals' performance. In trials where the sequence was reproduced correctly, the temporal patterns were different form incorrect ones. A second group of neurons was classified as *fixation cells*. These cells changed their activation in phase two of the experiment during fixation of a target before pressing it. Again the majority of them were selective for one specific target. These cells were reported to be primarily related to visual attention. The activation started when attention was directed to the target and after the target-pressing the unspecific activity was restored. The largest group of neurons was classified as *context cells*. Again the activation of the majority of the cells depended on the target on which the animal directed its attention. In addition different groups of these neurons encoded in which particular sequence the targets were activated. Thus, the activation was modulated by the context of future and past targets. Different classes of these cells have been identified being either highly selective on a specific sequence or encoding multiple sequences. This class of neurons is of particular interest in our discussion, because the cells encode hidden cause information. They infer information from the input that is not directly observable, but must be restored form the, usually noise activation of other cells. These findings have shown that the prefrontal cortex comprises divers classes of neurons related to memorising sequences, sequence decoding and action planning. Most of these cells were specific on one special aspect of the spatio-temporal input and encoded the sequential information on different levels of abstraction.

In a related study Averbeck and collaborators have shown that sequences of motor actions are represented by patterns of neuronal ensemble activity in prefrontal cortex in macaque monkeys [Averbeck et al., 2002]. The monkeys were trained to reproduce simple geometric shapes using a joystick-controlled cursor. Each trial started with a short waiting time after which the shape was displayed on a screen. If the shape was correctly reproduced the monkey was given a juice reward. The same shape was presented several times in successive trials, such that the monkey was aware of the shape during the waiting time after the first trial. Aside the neural recordings form prefrontal cortex the position of the joystick and the eye movements were monitored. The authors investigated the correlation between the motor trajectories and the neural response. Each drawing of a shape was composed of distinct movement segments and it was shown that these segments can be associated with patterns of neuronal ensemble activity. These activity patterns differed significantly among each other but on execution of the movement segment they were clearly represented by the neural readout. Notably, the activity patterns were also represented during the waiting phase and the strength of their representation was correlated with their serial order in the movement sequence. From this finding Averbeck et. al. concluded that '*the strength of segment representation is the neural code for serial order*' [Averbeck et al., 2002].

Yet a more abstract neural code for sequential order was found by [Berdyyeva and Olson, 2009]. They studied neurons form the supplementary eye field of rhesus monkeys, while the monkeys were performing different sequential eye movement tasks. It was already known from previous studies that during such tasks, some neurons from the supplementary eye field, encode the rank order of movements (see [Berdyyeva and Olson, 2009] and referenced therein). The authors investigated whether these neurons generalise the abstract information of rank order independent of the particular movement task. To address this issue they monitored the neural activity during serial action and a serial object tasks. The serial action tasks were designed such that the animals had to perform a series of eye movements in a learned sequence. In the serial object tasks the animals had to direct their gaze to objects appearing on random positions, in a learned sequences of object identities. Berdyyeva and Olson showed that a group of the studied cells, called *rank-selective neurons*, encoded solely the sequential order of the current movement within the sequence, independent of the task. They further showed that this behaviour is neither modulated by temporal aspects of the experiment nor reward expectation. Thus rank-selective neurons really encode the abstract information of rank positions within a sequence.

Based on these findings [Salinas, 2009] developed a neural model for memorising and replaying sequential data. They proposed a two layer, feed forward network: The input layer is composed of rank-order neurons as defined in the previous paragraph, the output layer consists of simple linear neurons which drive the motor units. As proposed by Berdyyeva and Olson, the rank-order neurons are activated in a fixed sequential order. Each rank in the sequence is represented by multiple neurons. This sequential pattern drives the output neurons to produce an action sequence. Multiple action sequences can be trained into the weights between input and output layer. To do so Salinas used different rate modulations for the rank-order neurons. For each action sequence a distinct pattern of firing rates was used for training and for pattern replay. These activity patterns were chosen at random, but needed to be constant over the whole action pattern. The replay was reported to be very robust to noise and the number of action patterns, that could be stored, was only limited by the number of rank-order neurons. One issue that is still open is how the rate modulation of the rank-order neurons is realised in a biological network. Not only that demanding firing rates of rank-order neurons to be modulated with the sequence identity is against the assumption that they encode information that is independent of any sequence identity, the neural mechanism of the modulation is unclear.

2.2 Neural replay of complex patterns

In the previous section we have seen that the prefrontal cortex plays an important role in memorising and planning sequential tasks. We pointed out that, when performing motor planning tasks, highly complex and abstract memory traces are replayed in that area of the brain. In this section we will review work, that has explored in more detail, how these memory traces are formed. The standard model of system consolidation suggests that recent memory traces are kept in the hippocampus for several days. During that time the memory is slowly moved to the neo-cortex where it is stored and can be recalled for several years or even live time [Roediger et al., 2007]. Experimental data suggests that this shifting of memory between two brain areas is accomplish by repeated replay of the memory traces during sleep. The prefrontal cortex has been under extensive research in this context.

The hippocampus formation and the neo-cortex are strongly connected. The main afferent axons originate in the entorhinal cortex, a subregion of the temporal lobe. Efferent connections are sent back to the entorhinal cortex and directly to the prefrontal cortex [Buzsáki, 1989]. The efferent connections give rise to cortical memory replay. Neural activity patterns in neo-cortex and hippocampus have shown to be densely coupled during slow wave sleep [Sirota et al., 2003]. Patterns that are generated when performing tasks in the awake state are replayed during this phase [Peyrache et al., 2009,Isomura et al., 2006]. In rat's prefrontal cortex this replay occurs in transient episodes that correspond to activity of distinct cell assemblies. It was also found that the patterns were most often replayed, when they had appeared in a situation of decision making in a learning task [Peyrache et al., 2009]. The pattern replay does not occur at the same speed as in the awake state. Patterns are compressed in time by a factor of about 6-7 [Euston et al., 2007]. The afferent connections from the neo-cortex to the hippocampus have been hypothesize to select unique subpopulations of hippocampal neurons [Isomura et al., 2006]. This allows to selectively replay hippocampal activity pattern during the formation of neocortical memory traces.

2.3 Implicit and statistical learning

Besides these studies on neural readouts from monkeys and rats, data gathered from behavioural studies on humans exists, that allows at least some insight into the underlying neural processes. Two major fields that are closely related are the paradigms of statistical learning and implicit learning. Both fields account for the same phenomenon - learning sequential data without clear awareness or even the intention to learn, just by being exposed to a rule-governed stimuli - but they use different methods to describe this phenomenon [Perruchet and Pacton, 2006]. *Statistical learning* was introduced by Saffran and collaborators [Saffran et al., 1996]. They reported that 8-month-old infants are able to segment an ongoing sequence of monotonic speech into words. This ability is not innate but can be learned by presenting a speech stream of nonsense words in random order. Since there were no pauses or other clues for the word boundaries, Saffran proposed that the word boundaries were inferred only from the statistics of the syllable transitions. The infants showed a significant discrimination between words and non-word stimuli after only two minutes of training.

Implicit learning was introduced in [Reber, 1967] and has since then been investigated by several authors (see [Cleeremans et al., 1998] for a review). A common experimental paradigm used in this field is *artificial grammar learning* (AGL). In an AGL task sequences of symbols are presented that have been generated by a finite-state grammar. After a memorisation phase, the subjects are asked to classify new sequences as being created by the same grammar or not in a forced-choice-task. Typically subjects perform better than chance would predict, but report that they were guessing and are unable to verbalise the rules of the grammar. Conway and Pisoni reported that the performance in these tasks depends on the modality used to present the data [Conway and Pisoni, 2008, Conway and Christiansen, 2005]. Tactile, visual and auditory stimulations were used. For tactile stimuli vibrotactile pulses were delivered to participants fingers, where each finger was associated with a different symbol. For visual stimuli, black squares appearing at different spatial locations on a screen were presented and the auditory stimuli were composed out of tone patterns. It was shown that the performance in the classification task was highest for the auditory modality 75% and about 62% for the other two modalities. They also showed that different grammars presented in parallel with different modalities could be learned relatively independent of each other. From these findings Conway and Pisoni concluded that although learning mechanisms share common aspects across modalities, the learning takes place in different modality specific areas.

2.4 Neural circuits of songbirds

Songbirds have impressive motor and memory abilities that have been subject to extensive research (see [Hahnloser and Kotowicz, 2010] for a recent review). The ability to sing is not innate, but learned in a similar way speech is acquired by human children. The songs are usually learned from an adult tutor, often the birds father. The learning involves two phases, a period of memorisation in which the vocal information is stored in long-term memory, and a sensorimotor phase in which the birds own song is compared and refined to the memorised information [Bolhuis and Gahr, 2006]. In the first phase a '*template*' of the tutor song is stored in long-term memory. When the birds start to vocalise in the second phase, the vocal repertoire is still restricted. These first vocalisations are called subsongs and are considered to be the avian counterpart to human babbling [Aronov et al., 2008]. In this phase the bird matches its own song with the memorised ones, refining its singing abilities as well as the tutor template. It is believed that the bird uses the auditory template to produce a feedback as an error-correction mechanism for its own song. This theory is supported by the fact that birds need to hear themselves to learn their song [Konishi, 1965]. After this training phase, the produced songs of the adult bird resemble the tutor songs.

The exact locations and neural mechanisms of song learning are still unknown. Traditionally, a set of specialised brain areas, known as the song-control system were believed to be the neural substrate for song production and learning. The system was separated into a motor pathway responsible for song production and a basal-ganglia pathway for learning [Vates and Nottebohm, 1995]. This model was proposed because lesions in the basal-ganglia pathway impaired the birds of learning songs, but did not affect the production of songs already learned. However, this simplistic view was not sufficient to explain data from more recent studies, which have shown that the inability to learn resulted in fact from motor deficit. It was also shown that subsongs were generated by the basal-ganglia pathway and not the motor pathway, suggesting that both pathways carry motor information [Aronov et al., 2008].

Also brain regions outside the traditional song-control system were found to be involved in song learning. The caudomedial nidopallium (NCM) and caudomedial mesopallium (CMM) receive input from the lower auditory fields. The two areas have bidirectional connections and NCM is connected to the song-control system. They are often viewed as being analogous to Wernickes area in humans [Hahnloser and Kotowicz, 2010]. Increased gene expression in NCM and CMM was observed when birds were exposed to conspecific songs, but much less when exposed to heterospecific songs. Repeated playbacks of one song resulted in increased gene expression in NCM for the first 30 minutes only, thus an increased response to novelty. Singing of deafened birds showed no increased response. Areas NCM and CMM are thus assumed to be related to auditory memory and as being part of the neural substrate of the tutor template [Bolhuis and Gahr, 2006].

3 Probabilistic Models

Probabilistic models are powerful tools extensively used in machine learning. They allow to describe complex problems in a general framework, and to include prior knowledge and uncertainty easily. In this chapter we will review algorithms and models that are used throughout this thesis. A detailed descriptions of the concepts introduced here can be found in [Bishop, 2006].

3.1 Graphical models and hidden causes

Probabilistic models define the relationships between a set of random variables. The value of a random variable is unknown, but a probability distribution over all possible values can be defined. In many cases, the knowledge about the outcome of one random variable may change the distribution over the outcomes of another variable. For example, observing that the street is wet increases the probability that it has recently rained. This causal relationship between the two random variables (*the street is wet*) and (*it has recently rained*) can be described by a graphical representation shown in figure 3.1, known as Bayesian network. A Bayesian network defines random variables and their relationships. Here, observed variables are represented by blue, hidden ones by white nodes. In our simple example the observed variable x corresponds to *the street is wet*, the hidden variable z corresponds to *it has recently rained*. z is called a hidden variable because it can not be directly observed (in the example,

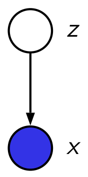


Figure 3.1: The Bayesian network of an observed variable and its hidden cause. Observed variables are represented by blue, hidden ones by white node. Relationships between variables are indicated by arrows.

because it lies in the past), but it has some effect on another variable that can be observed. This relationship between the random variables is represented by a parent-child relation, indicated by an arrow from the parent to the child. The observed and the hidden variables form a unit which is usually referred to as the *complete data*, whereas if the outcomes of the hidden variables are unknown one speaks about *incomplete data*.

The joint distribution of a Bayesian network is given by the product of the probability distribution of all random variables conditioned on their direct parents. Thus, the joint distribution of our example Bayesian network is given by

$$p(x, z) = p(z)\, p(x \mid z) \tag{3.1}$$

In this simple example both random variables x and z are binary. Inference about the hidden variable can be done easily by probability calculus. For example the probability that it has rained, given that the street is wet is given by

$$p(z = 1 \mid x = 1) = \frac{p(z = 1)\, p(x = 1 \mid z = 1)}{p(z = 0)\, p(x = 1 \mid z = 0) + p(z = 1)\, p(x = 1 \mid z = 1)} \tag{3.2}$$

In the next section we will discuss how the probability tables on the right hand of equation (3.2) can be learned in a general framework.

3.2 The expectation-maximisation algorithm

The parametrisation of the simple example model from section 3.1 was given by the prior and conditional probabilities, $p(z)$ and $p(x \mid z)$ respectively, which shall be collected in the parameter vector θ. In a general learning task a sequence of T samples from the observed variable $\mathbf{X} = \{x^{(1)} \dots x^{(T)}\}$ is given. The problem of estimating the optimal model parameters from a given observation sequence $\mathbf{X}$ is solved by maximising the likelihood function

$$\mathcal{L}(\theta \mid \mathbf{X}) = p(\mathbf{X} \mid \theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z} \mid \theta). \tag{3.3}$$

The sum over $\mathbf{Z}$ denotes the sum over all possible outcomes of the sequences of hidden variables. Unfortunately there exists no closed form solution to solve this maximisation problem. The problem arises because the data set is incomplete, since the hidden variables are unknown, which requires to consider all possible hidden state configurations. The standard algorithm for the maximisation of likelihoods with incomplete data is the *expectation-maximisation* (EM) algorithm [Dempster et al., 1977]. It finds a local maximum of $\mathcal{L}(\theta \mid \mathbf{X})$ by applying an iterative update rule. In every iteration, the algorithm replaces the current parameter set θ^{old} by a new one θ^{new} that is guaranteed to improve the model. Each iteration consists of two steps, the expectation (E) and maximisation (M) step. In the E step an expectation of the complete data log likelihood is evaluated based on the conditional probability $p\left(\mathbf{Z} \mid \mathbf{X}, \theta^{old}\right)$. Using this, an auxiliary function of the old and the new parameter set is defined, given by

$$Q\left(\theta, \theta^{old}\right) = \sum_{\mathbf{Z}} p\left(\mathbf{Z} \mid \mathbf{X}, \theta^{old}\right) \log p\left(\mathbf{Z}, \mathbf{X} \mid \theta\right). \tag{3.4}$$

In the M step a parameter set θ^{new} is found that maximises Q and the old parameter set is replaced by the new one. This procedure can be proven to increase $\mathcal{L}(\theta|\mathbf{X})$ monotonically until some local optimum was reached.

3.3 Monte Carlo methods

Monte Carlo methods can be used to solve two problems [MacKay, 2003]:

1. Generate a set of L independent samples from a probability distribution $p(z)$.

2. Estimate the expectation of a function under a given distribution.

Both problems are solved using numerical sampling. The basic idea of Monte Carlo methods is that the expectation of a function $f(z)$ with respect to a probability distribution $p(z)$ given by

$$\mathbb{E}\{f\} = \int f(z)\, p(z)\, \mathrm{d}\, z, \tag{3.5}$$

can be approximated by a finite sum over a set of L independent samples $z_1 \ldots z_L$ drawn from the distribution $p(z)$, given by

$$\hat{f} = \frac{1}{L} \sum_{l=1}^{L} f(z_l). \tag{3.6}$$

As L converges to infinity $\hat{f}$ will become identical to the expected value (3.5). Monte Carlo methods are of particular interest, when $p(z)$ is too complex to evaluate it directly. There exist two major classes of Monte Carlo methods: one that is based on importance sampling and Markov chain Monte Carlo techniques [Hernández et al., 1996].

3.3.1 Importance sampling

If drawing from $p(z)$ directly is not possible, *importance sampling* can be used to approximate equation (3.5). Let us assume that the distribution $p(z)$ is known and can be evaluated up to a normalising constant, but it is too complex to draw samples from it easily. Then, equation (3.5) can be approximated by drawing samples $z_1 \ldots z_L$ from a *proposal distribution* $q(z)$. The proposal distribution can be any distribution that is non-zero wherever $p(z)$ is non-zero. Equation (3.6) is then rewritten

$$\hat{f} = \frac{1}{L} \sum_{l=1}^{L} r_l \cdot f(z_l), \qquad \text{with:} \quad r_l = \frac{p(z_l)}{q(z_l)}. \tag{3.7}$$

The factors r_l are called *importance weights* and correct the bias introduced by sampling from the wrong distribution. In general it is desirable that the proposal is as close as possible to the true distribution, such that the importance weights are close to a constant value [Bishop, 2006].

In a related approach, known as *rejection sampling* the importance weights are given by binary random variables $r_l \in \{0, 1\}$, thus the sample is either accepted or complete rejected from the sum (3.7). The probability that a sample is accepted is given by

$$p(r_l = 1) = c \cdot \frac{p(z_l)}{q(z_l)}, \tag{3.8}$$

where c is a constant that should be as large as possible under the constraint that $p(r_l = 1) \leq 1$ such that, as little samples are rejected as possible.

3.3.2 Markov chain Monte Carlo

So far, we considered samples that are drawn independently from a proposal distribution, for the approximate expectation (3.7). In order to converge fast to the target function (3.5) we require that a large number of drawn samples lies within regions with high probability in the target density $p(z)$. Unfortunately this requirement is hard to achieve with importance sampling approaches for high dimensional problems [MacKay, 2003]. More powerful sampling techniques, that scales well with dimensionality are methods based on Markov chain Monte Carlo (MCMC). They are similar to rejection sampling, with the major difference that samples are not drawn independently. Instead a proposal distribution $q(z_l \mid z_{l-1})$ is defined that depends on the last sample that has been excepted z_{l-1}. If a new sample is excepted z_l is assigned to that new value, if it is rejected z_l is assigned to the previous value z_{l-1}. The sequence of samples $z_1 \dots z_L$ forms a Markov chain, which increases the chance that many samples are drawn from dense regions of the target distribution. Using MCMC, the probability space is faster explored which results in more efficient and accurate sampling methods. In order to obtain independent samples using MCMC methods only every M^{th} sample is retained. For M sufficiently large the samples can be considered independent for all practical purposes [Bishop, 2006].

3.3.3 Monte Carlo EM

Monte Carlo methods are of particular interest here, because they can be used to approximate the E step of the EM algorithm, by drawing a finite set of hidden variables given the current model parameters. More precisely, equation (3.4) can be approximated by a finite sum over L samples drawn from the posterior distribution $p\left(\mathbf{Z} \mid \mathbf{X}, \theta^{old}\right)$, give by

$$Q\left(\theta, \theta^{old}\right) \approx \frac{1}{L}\sum_{l=1}^{L} \log p\left(\mathbf{Z}_l, \mathbf{X} \mid \theta\right). \tag{3.9}$$

The approximate Q function is then optimised using the exact M step. This procedure is called *Monte Carlo EM* [Bishop, 2006].

The special case, where only a single sample is drawn from the posterior distribution is called *stochastic EM*. The sample drawn in the E-step completes the data set. Thus, in the M-step a complete data maximum likelihood estimation is performed, which is easy to solve for many practical problems [Nielsen, 2000].

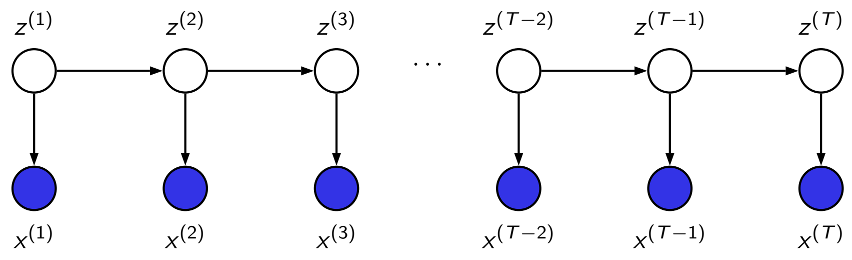


Figure 3.2: The Bayesian network of the HMM. The state sequence forms a Markov chain on top of the observation sequence. There is no direct dependence between the observations, all dependences are covered by the state sequence.

3.4 The hidden Markov model

In section 3.1 we introduced a simple example model with a hidden and an observed variable. We showed that by using the EM algorithm, we can learn the parametrisation of such models, given a sequence of observed variables only. Yet, the calculations we used, did not capture any temporal structure of that sequence. Thus, presenting the sequence in any random order would lead to similar results. In practice however, successive samples from natural sources will be correlated and exploiting this correlations will significantly increase the descriptive power of probabilistic models. The *hidden Markov model* captures these temporal dependences of successive samples by introducing dependences between the hidden cause variables. In this section we will give a short introduction on the theory of discrete HMMs, for a complete review see [Rabiner, 1989, Bengio, 1999, Bishop, 2006, Yamagishi, 2006].

The hidden Markov model is a statistical time series model for describing a time varying signal $\mathbf{X} = \{x^{(1)} \dots x^{(T)}\}$. It assumes that the observed sequence has been created by an underlying, unobserved process. This process is modelled by a probabilistic state machine with K states. Each state encodes a probability distribution over the observations and in each time step a transition between states according to a table of transition probabilities is made. Thus the HMM assigns a sequence of latent state variables $\mathbf{Z} = \{z^{(1)} \dots z^{(T)}\}$ to the sequence of observations $\mathbf{X}$. The temporal dependence between time instances is captured in the state variables. The state sequence can be described by a Markov chain which is illustrated by the Bayesian network in figure 3.2. One can see the Markov chain of state transitions on top of the observations.

By exploring the structure of the Bayesian network many useful independence properties can be discovered. One of the most important ones is that the whole temporal information within $\mathbf{X}$ is covered by the Markov process. If the state variables are given, the observations become independent. The second independence property, known as the *Markov property*, states that given a hidden cause $z^{(t)}$ at any time point t, all observations before t become independent of all observations after t. We will make extensive use of these properties throughout our discussions.

We will only consider HMMs with discrete states and discrete time. Throughout this thesis we will use the following notation: Each state at time t is represented by a binary vector $\mathbf{z}^{(t)} \in \{0,1\}^K$, with K elements $z_k^{(t)}, k \in \{1 \dots K\}$. The vector elements obey the constraint $\sum_{k=1}^{K} z_k^{(t)} = 1, \forall t$. The model being in the k^{th} state at time t is then represented by setting $z_k^{(t)}$ to one. We will use similar considerations for the input variable. Let us consider N discrete observations which can be collected in a binary vector $\mathbf{x}^{(t)} \in \{0,1\}^N$ with elements $x_i^{(t)}$, that follow the same constraint as the state vectors. In addition, we use the shorthand notation $p\left(z_k^{(t)}\right)$ for $p\left(z_k^{(t)} = 1\right)$.

3.4.1 HMM parametrisation

A HMM is characterised by its set of parameters $\theta = \{\mathbf{A}, \mathbf{B}, \pi\}$, where $\mathbf{A}$ is the table of *transition probabilities*, $\mathbf{B}$ is the table of *observation probabilities*, the probability of seeing an observation in a certain state and π are the *prior probabilities* for the initial states. Let us denote the elements of these probability tables by a_{kj}, b_{ki} and π_k given by

$$\begin{aligned} a_{kj} &\equiv p\left(z_k^{(t)} \mid z_j^{(t-1)}\right), \\ b_{ki} &\equiv p\left(x_i^{(t)} \mid z_k^{(t)}\right), \\ \pi_k &\equiv p\left(z_k^{(1)}\right). \end{aligned} \tag{3.10}$$

Here, we only consider *homogeneous models* for which the set of parameters is independent of time.

There are three basic problems to be solved for HMMs, for each of which an efficient algorithm exists:

- Calculate the probability of an observation sequence $\mathbf{X}$ given the model $p(\mathbf{X} \mid \theta)$.

- Given an observation sequence $\mathbf{X}$ find the most probable path, which is usually referred to as *Viterbi path*.

- Adjust the model parameters θ to a given observation sequence $\mathbf{X}$.

All solutions to these problems make extensive use of the independence properties of the HMM. Of particular interest is that the joint probability over the complete data factorises into the form

$$\begin{aligned} p\left(\mathbf{X},\mathbf{Z} \mid \theta\right) &= p\left(\mathbf{z}^{(1)}\right) p\left(\mathbf{x}^{(1)} \mid \mathbf{z}^{(1)}\right) \prod_{t=2}^{T} p\left(\mathbf{z}^{(t)} \mid \mathbf{z}^{(t-1)}\right) p\left(\mathbf{x}^{(t)} \mid \mathbf{z}^{(t)}\right) \\ &= \pi_{\mathbf{z}^{(1)}} \cdot b_{\mathbf{z}^{(1)},\mathbf{x}^{(1)}} \prod_{t=2}^{T} a_{\mathbf{z}^{(t)},\mathbf{z}^{(t-1)}} \cdot b_{\mathbf{z}^{(t)},\mathbf{x}^{(t)}} \end{aligned} \tag{3.11}$$

In the next section we will present an efficient solution for problem three based on this factorisation.

3.4.2 The Baum-Welch algorithm

The standard algorithm to find the HMM parameters θ^* that maximises the incomplete data likelihood (3.3) is the *Baum-Welch algorithm* [Baum and Petrie, 1966]. The Baum-Welch algorithm is an instance of the EM algorithm. We will review here the basic Baum-Welch equations in more detail.

In the standard HMM literature, in order to introduce the E step for HMMs, a short hand notation for the probability of making a transition from state $j \rightarrow k$ and the probability of being in state k at time t, given an observation $\mathbf{X}$, is introduced

$$\begin{aligned} \xi\left(z_k^{(t)}, z_j^{(t-1)}\right) &= p\left(z_k^{(t)}, z_j^{(t-1)} \mid \mathbf{X}, \theta\right), \\ \gamma\left(z_k^{(t)}\right) &= p\left(z_k^{(t)} \mid \mathbf{X}, \theta\right) = \sum_{l=1}^{K} \xi\left(z_k^{(t)}, z_l^{(t-1)}\right). \end{aligned} \tag{3.12}$$

Using these quantities we can define the set of parameters that maximises the

Q function (3.4) which can be found using Lagrange multipliers method

$$\begin{aligned} a_{kj}^{new} &= \frac{\sum_{\tau=1}^{T} \xi\left(z_k^{(\tau)}, z_j^{(\tau-1)}\right)}{\sum_{\tau=1}^{T} \gamma\left(z_j^{(\tau-1)}\right)}, \\ b_{ki}^{new} &= \frac{\sum_{\tau=1}^{T} \gamma\left(z_k^{(\tau)}\right) x_i^{(\tau)}}{\sum_{\tau=1}^{T} \gamma\left(z_k^{(\tau)}\right)}, \\ \pi_k^{new} &= \frac{\gamma\left(z_k^{(1)}\right)}{\sum_l \gamma\left(z_l^{(1)}\right)}. \end{aligned} \tag{3.13}$$

These formulas have a nice frequentist interpretation: The sum over the whole state sequence over ξ yields the sufficient statistics of the HMM. We can define

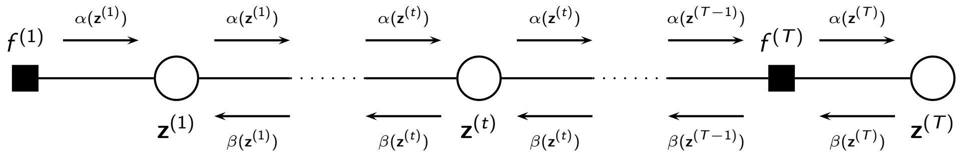
$$n_{kj} = \sum_{\tau=1}^{T} \xi\left(z_k^{(\tau)}, z_j^{(\tau-1)}\right) \quad \text{and} \quad n_k = \sum_{\tau=1}^{T} \gamma\left(z_k^{(\tau)}\right) \tag{3.14}$$

as the expected number of $z_j \to z_k$ transitions and expected number of being in state k, respectively. The expectation is taken over the whole input sequence. The update equation for the transition probabilities is just a fraction of these two expected numbers. Further note that all update equations (3.13) can be written in terms of ξ and thus, evaluation of this quantity is the central task of the E step. In the next section we will introduce an efficient algorithm for this task.

3.4.3 The forward-backward algorithm

By inspecting the independence properties of the Bayesian network in figure 3.2 we see that we can rewrite equation (3.12) as

$$\begin{aligned} \xi\left(z_k^{(t)}, z_j^{(t-1)}\right) &= p\left(z_k^{(t)}, z_j^{(t-1)} \mid \mathbf{X}\right) \\ &= \frac{p\left(\mathbf{X} \mid z_k^{(t)}, z_j^{(t-1)}\right) p\left(z_k^{(t)}, z_j^{(t-1)}\right)}{p\left(\mathbf{X}\right)} \\ &= \frac{p\left(\mathbf{X}^{(1 \ldots t-1)}, z_j^{(t-1)}\right) p\left(\mathbf{x}^{(t)} \mid z_k^{(t)}\right) p\left(z_k^{(t)} \mid z_j^{(t-t)}\right) p\left(\mathbf{X}^{(t+1 \ldots T)} \mid z_k^{(t)}\right)}{p\left(\mathbf{X}\right)} \end{aligned}$$



$f^{(1)}$
$\alpha(\mathbf{z}^{(1)})$
$\alpha(\mathbf{z}^{(1)})$
$\alpha(\mathbf{z}^{(t)})$
$\alpha(\mathbf{z}^{(t)})$
$\alpha(\mathbf{z}^{(T-1)})$
$f^{(T)}$
$\alpha(\mathbf{z}^{(T)})$
$\mathbf{z}^{(1)}$
$\beta(\mathbf{z}^{(1)})$
$\beta(\mathbf{z}^{(t)})$
$\mathbf{z}^{(t)}$
$\beta(\mathbf{z}^{(t)})$
$\beta(\mathbf{z}^{(T-1)})$
$\beta(\mathbf{z}^{(T)})$
$\mathbf{z}^{(T)}$

Figure 3.3: Illustration of the message passing in the forward-backward algorithm. The factor graph of the HMM is shown and the flow of messages is indicated. Since $\mathbf{X}$ is given the graph can be simplified by absorbing the emission probabilities into the factors, which are then given by $f^{(1)}\left(\mathbf{z}^{(1)}\right) = p\left(\mathbf{z}^{(1)}\right) p\left(\mathbf{x}^{(1)} \mid \mathbf{z}^{(1)}\right)$ and $f^{(t)}\left(\mathbf{z}^{(t-1)}, \mathbf{z}^{(t)}\right) = p\left(\mathbf{z}^{(t)} \mid \mathbf{z}^{(t-1)}\right) p\left(\mathbf{x}^{(t)} \mid \mathbf{z}^{(t)}\right)$.

where we have dropped the dependence on θ assumed to be constant. This expression can be calculated efficiently by the *forward-backward* algorithm. The algorithm consists of two recursive equations, one running forward and one backward in time given by

$$\begin{aligned}\alpha\left(\mathbf{z}^{(t)}\right) &= p\left(\mathbf{X}^{(1\ldots t)}, \mathbf{z}^{(t)}\right) \\ &= \sum_{\mathbf{z}^{(t-1)}} p\left(\mathbf{x}^{(t)} \mid \mathbf{z}^{(t)}\right) p\left(\mathbf{z}^{(t)} \mid \mathbf{z}^{(t-1)}\right) \alpha\left(\mathbf{z}^{(t-1)}\right) \end{aligned} \qquad (3.15)$$

and

$$\begin{aligned}\beta\left(\mathbf{z}^{(t)}\right) &= p\left(\mathbf{X}^{(t+1\ldots T)} \mid \mathbf{z}^{(t)}\right) \\ &= \sum_{\mathbf{z}^{(t+1)}} p\left(\mathbf{x}^{(t+1)} \mid \mathbf{z}^{(t+1)}\right) p\left(\mathbf{z}^{(t+1)} \mid \mathbf{z}^{(t)}\right) \beta\left(\mathbf{z}^{(t+1)}\right). \end{aligned} \qquad (3.16)$$

Inserting this into equation (3.12) we can write

$$\xi\left(z_k^{(t)}, z_j^{(t-1)}\right) = \frac{\alpha\left(z_j^{(t-1)}\right) p\left(\mathbf{x}^{(t)} \mid z_k^{(t)}\right) p\left(z_k^{(t)} \mid z_j^{(t-t)}\right) \beta\left(z_k^{(t)}\right)}{p\left(\mathbf{X}\right)} \qquad (3.17)$$

In the E Step these equations are evaluated in terms of the current HMM model parameters given in equation (3.10). The normalisation in equation (3.17) is usually omitted since it cancels out when calculating the parameter update (3.13). The forward messages start running at the first and the backward messages at the final time step T. The messages are initialised to be

$$\alpha\left(z_k^{(1)}\right) = p\left(z_k^{(1)}\right) p\left(\mathbf{x}^{(t)} \mid z_k^{(1)}\right) \quad \text{and} \quad \beta\left(z_k^{(T)}\right) = 1. \qquad (3.18)$$

To get an intuitive interpretation of this algorithm consider figure 3.3. The factor graph of the HMM and the flow of forward and backward information is shown. The forward messages start at the first factor and are passed on forward in time until they reach the last node. The backward messages start at the last variable node and are passed back to the first one. At each factor node they pass, the messages are updated using equations (3.15) and (3.16). At each variable node the messages are passed through unchanged. Thus, the forward-backward algorithm is an instance of the *sum-product* algorithm and the calculation of the forward and backward pass is equivalent to the message passing in that framework [Bishop, 2006]. Note that for evaluating equation (3.17) at each time step, each message only needs to be computed once and thus this procedure can be done efficiently.

3.5 Extensions of the basic HMM

Since its introduction by [Baum and Petrie, 1966] the HMM has been under extensive research. Today, a huge collection of extensions and simplifications exist for the basic HMM equations. In this section we give a brief overview of the work, that is most relevant for this thesis.

3.5.1 Online HMM learning

In a real world learning task the HMM is not adapted to a single observation sequence $\mathbf{X}$, but to a whole batch of sequences $\mathcal{X} = \{\mathbf{X}_1 \dots \mathbf{X}_S\}$. For example each of the sequences might be features extracted from a certain word spoken by one of S different speakers to learn a HMM word model. To achieve this using the Baum-Welch algorithm the forward-backward algorithm is run for each sequence separately to give $\xi_{kjs}^{(t)}$, $s \in \{1 \dots S\}$ and then averaged over the whole batch

$$\xi_{kj}^{(t)} = \frac{1}{S} \sum_{s=1}^{S} \xi_{kjs}^{(t)}, \tag{3.19}$$

from which the M step is then computed [Rabiner, 1989]. We will refer to this approach as *batch learning*.

Obviously this requires to store the whole batch of training sequences. Since their number can grow very large in certain tasks, online algorithms are of increasing interests. In online learning the whole batch is not assumed to be given, but pieces of the batch arrive one after another. In the HMM literature there are two different definitions of how this arrival of new data takes place. In [Baldi and Chauvin, 1994] each of the sequences $\mathbf{X}_l$ is assumed to arrive one after another. The batch of sequences is enlarged, but the sequences arrive as a whole. The growing amount of data renders batch Baum-Welch learning infeasible, but the forward-backward calculations can still be evaluated on each new sequence. The second definition was used in [Mongillo and Deneve, 2008, Stiller and Radons, 1999]. The input is assumed to be given by a single sequence that is lengthened by appending a new sample $\mathbf{x}^{(t)}$ to the end of the sequence at each time step. From the definition of the forward-backward message passing one can easily see, that when a new sample arrives all backward messages need to be updated, which makes the forward-backward algorithm infeasible for increasing sequence lengths. In order not to confuse these different approaches we will use the following nomenclature: We refer to the first approach as *online learning* and to the latter one as *incremental learning*. In addition, we always refer with *sample* to a single observation $\mathbf{x}^{(t)}$ of a sequence of observations.

An online HMM learning approach was given by [Baldi and Chauvin, 1994]. The HMM parameters are given by a normalized exponential representation for which a smooth gradient-based learn rule can be derived. For the weight update the standard forward-backward equations are computed on a single sequence. The authors showed that the algorithm converges to the same result as the Baum-Welch algorithm. This work is very similar to the approach that will be introduced in the original part of this thesis. For the incremental HMM learning in [Mongillo and Deneve, 2008, Stiller and Radons, 1999] a update rule for the HMM's sufficient statistics was derived. The sufficient statistics are kept in a Parameter Tensor and updated incrementally for each arriving sample. The equations were proven to be equivalent to Baum-Welch algorithm. This approach does not require to pass information backwards in time directly. The backward pass of information is achieved by making the model parameters explicitly dependent on the current time step. The parameters can be updated incrementally, but with higher computational complexity.

3.5.2 Simplifications and extensions of the Baum-Welch algorithm

The segmental k-means algorithm [Juang and Rabiner, 1990] approximates the Baum-Welch equations by using only the most likely path sequence (Viterbi path). The most likely path sequence can be efficiently calculated using the

Viterbi algorithm [Viterbi, 1967], which has a similar structure to the forward-backward algorithm. Thus, the HMM parameters are estimated using a single path. The training results obtained by the segmental k-means algorithm are similar to that calculated using standard Baum-Welch. In [Merhav and Ephraim, 1991] a upper bound between the segmental k-means and the Baum-Welch likelihoods has been proven.

In [Alamino and Caticha, 2008] the HMM parameters are updated using a Bayesian learning approach, assuming a Dirichlet prior over the parameters. The exact calculation requires to sum over all possible path sequences and are thus intractable for practical cases. A Mean Field approximation for Gaussian distributions was shown. In [Huda et al., 2009] a hybrid between standard and stochastic EM was introduced to overcome local minima. The contribution of stochastic EM was controlled using the Simulated Annealing technique. The authors reported that the proposed algorithm outperformed standard EM in a speech processing task.

The HMM theory was generalised to continuous state space in [Thrun et al., 1999]. The Baum-Welch equations are adapted to continuous state space by using a sampling-approximation of the forward- and backward-messages using importance - sampling - resampling. The sampled distributions are then represented in density trees so that the Baum-Welch equations can be evaluated. In [Krishnamurthy and Moore, 1993] an online learning algorithm was derived for a HMM with Gussian distributed observations. The derivations are based on maximization of the Kullback-Leibler divergence and use the properties of distributions form the exponential family. But, they do not generalise to other distributions.

4 Neuron Models

Networks of biological neurons are highly non-linear structures that can show rich and very complex behaviour. Each neuron receives input from some hundred other cells, integrates this information and produces a discrete output, which leads to dynamics that are very hard to predict. The mathematical description of these dynamics has been a major issue in machine learning and computational neuroscience. First attempts to address this problem range back more than 60 years. Starting form these basic models various research directions have developed neural models with different levels of detail and complexity. In this section we will review some of the major results.

4.1 Standard neuron models

It is well accepted, that most computations in the nervous system are based on the electrical properties of neurons. Neurons are special cells that are able to control their *membrane potential*, the electric potential between the inter- and the extra-cellular liquid. The membrane potential is controlled by actively transporting ions through the cell membrane or letting them passively flow through specialised ion channels. Typically different types of ions are involved, each of which is gated through a specific ion channels. A typical neuron is composed of three functionally distinct parts: The *dendrites* over which it receives input from other neurons, the *soma* that integrates this information and generates the output that is propagated along the *axon* to downstream neurons [Gerstner and Kistler, 2002] (See figure 4.1 for a schematic representation). Most neurons of the mammalian brain form connections to other neurons through chemical synapses. With chemical synapses, the electric potential of the pre-synaptic neuron is not directly propagated to the post-synaptic neuron, but transmitted by expressing a neurotransmitter that is translated back to a membrane potential at the post-synaptic side. The strength of these connections is usually adaptive, which is considered to be the major mechanism of learning in the brain. Synapses can contribute a depolarising effect to the membrane potential, which are called excitatory synapses, or they pull

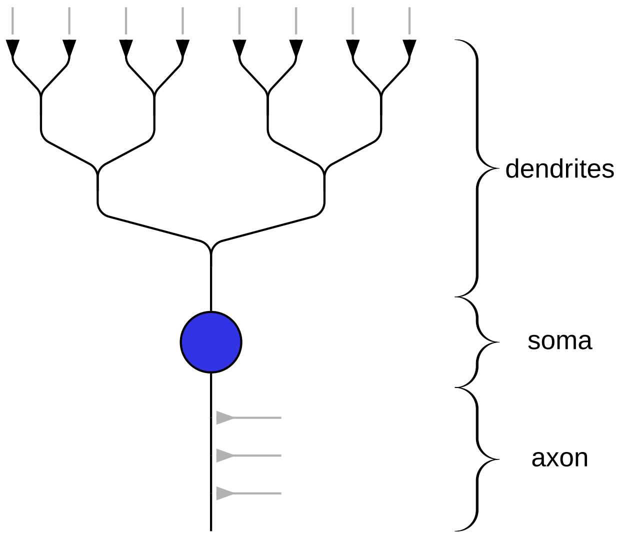


Figure 4.1: Schematic representation of a neuron. Most neurons of the brain are composed of three parts: The dendrites that receive synaptic connections from other neurons (synapses are indicated by small triangles), the soma and the axon, that projects to downstream neurons (downstream neuron's synapses indicated in gray).

the membrane potential closer to the cells resting potential, called inhibitory synapses. The sum of all synaptic currents is collected over the dendrites and propagated to the soma. If depolarisation at the soma exceeds some specific threshold a spike is generated that is transmitted over the axon and may be received by other downstream neuron's synapses.

4.1.1 The McCulloch and Pitts neuron

One of the first artificial neuron models that tried to model biological neurons was introduced in [McCulloch and Pitts, 1943]. All dynamics of the dendrites were reduced to calculating the membrane potential u at the soma by a weighted sum over the activation of N input neurons

$$u = \sum_{i=1}^{N} w_i x_i + w_0. \tag{4.1}$$

The weights w_i are the synaptic strengths to the i^{th} input neuron. The bias weight w_0 can be used to move the neuron's resting potential. The output z is computed by a simple threshold function with a threshold parameter ϑ

$$z = \begin{cases} 1 & \text{if } u \geq \vartheta \\ 0 & \text{else} \end{cases} . \tag{4.2}$$

Despite its strong simplifications compared to biological neurons, the McCulloch and Pitts neuron and its extensions have proven to be very powerful models. Efficient algorithms exists to adapt the network weights of single neurons or whole networks and once trained evaluation of the network breaks down to simple equations. Due to this, artificial neural networks have become and important field in machine learning [Bishop, 2006].

4.1.2 The leaky integrate-and-fire neuron

The leaky integrate-and-fire neuron is considered to be a good trade-off between biological accuracy and computational tractability and is probably the best-known formal spiking neuron model [Gerstner and Kistler, 2002]. The electrical circuit of the membrane at the soma is modelled by a capacitor C in parallel with a resistor R. The driving current I injected by the dendrites is split into a current that charges the capacitor and a leaky current discharging it over the resistor, modelling ion leakage through open channels. The membrane potential u is given by the voltage across the capacitor which follows the differential equation

$$RC\frac{\mathrm{d}u}{\mathrm{d}t} = -u^{(t)} + RI^{(t)} \tag{4.3}$$

An output spike is emitted whenever the membrane potential exceeds the threshold ϑ. After the spike was emitted the membrane potential is reset to the resting potential $u_r < \vartheta$ [Gerstner and Kistler, 2002].

While the spike generation of the soma is described accurately in this model, there are no assumptions made upon the driving current I. In a biological setup the driving current will by governed by the activity of the presynaptic neurons observed over the neuron's dendrites. Each spike received at a synapse will contribute a current pulse to I. The current that is injected by each synapse is proportional to its excitatory post synaptic potential (EPSP), the depolarisation induced by synaptic activity. The total input current is then given by the sum over all current pulses, weighted by the synaptic efficiencies

$$I^{(t)} = \sum_i w_i \sum_f \alpha_i \left(t - t_{i,f}\right), \tag{4.4}$$

where $t_{i,f}$ are lists of all input spike times arriving at synapse i. The kernel $\alpha_i\left(t\right)$ is the shape of the current pulse injected by synapse i, which is a Dirac δ-pulse in the simplest case. In a more realistic setup the EPSPs can be modelled by a double exponential kernel, with finite fall- and rise times τ_s and τ_r, respectively

$$\alpha_i\left(t\right) = \frac{q}{\tau_s - \tau_r}\left(exp\left(-\frac{t-\Delta_i}{\tau_s}\right) - exp\left(-\frac{t-\Delta_i}{\tau_r}\right)\right)\Theta\left(t-\Delta_i\right), \tag{4.5}$$

where $\Theta\left(t\right)$ is the Heaviside step function, q is the total charge that is injected via a single synapse and Δ_i are transition delays induced by the synapse's distance to the soma [Gerstner and Kistler, 2002].

4.2 Synaptic plasticity

So far we have only stated, that learning in the brain is achieved by changing the synaptic strength, but we did not give a formal rule or neural mechanism that allows a synapse to show this plastic behaviour. In this section we will close this gap. First we will review the classic theory of Hebbian learning and then more recent results on spike time dependent plasticity.

4.2.1 Hebbian learning

Short after the first artificial neural network models were introduced by McCulloch and Pitts, Hebb proposed his theory on correlation based learning [Hebb, 1949]. He investigated the ability of neural networks to self-organise from a theoretical point of view, and proposed a very simple rule for adapting the synaptic strengths:

'*When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*' [Hebb, 1949]

Thus, the synaptic strength between two cells, is increased when the pre-synaptic cell takes part in firing the post-synaptic cell. A more formal definition is given by the equation for the weight change Δw_{ki}, between a presynaptic cell i and the postsynaptic cell k. The weight change depends on the binary input x_i received from cell i and the binary output z_k of cell k

$$\Delta w_{ki} = \eta x_i z_k, \tag{4.6}$$

where η is a learn rate parameter. This is the basic Hebbian learn rule, the synaptic weights trained with this rule directly reflect the correlation between the pre- and post-synaptic neuron. However, the learn rule lacks biological plausibility since the weight change is always positive or zero and thus the weights will grow to infinity [Gerstner and Kistler, 2002].

4.2.2 Spike time dependent plasticity

The first experimental result that supported Hebbs theory on correlation-based learning was given by [Bliss and Lømo, 1973]. They recorded the membrane potentials of cells in the Hypocampus of the rabbit, when pre-synaptic cells were stimulated. They found that the synaptic efficiency of granule cell's synapses can be strengthened by tetanic stimulation of the pre-synaptic cell. The increase in synaptic efficiency lasted for several hours, a phenomenon that is now referred to as long term potentiation (LTP) and the opposite phenomenon, where synaptic efficiency is decreased as long term depression (LTD) [Dan and Poo, 2004]. These mechanisms can be understood as the biochemical implementation of Hebbian learning [Lisman, 1989]. Yet more recent studies have shown that not only the coincident activation, but the precise timings of pre- and post-synaptic spikes is crucial for synaptic plasticity. An increase in synaptic efficiency is only observed if pre-synaptic spikes arrive within a critical time window [Markram et al., 1997]. This phenomenon is referred to as spike time dependent plasticity (STDP) [Dan and Poo, 2004].

In general, the value of the synaptic efficiency change is a function of the pre- and post-synaptic spike times. In many cases, only the difference of a single spike pair is taken into account, but also combinations of multiple spike times have been considered [Froemke and Dan, 2002]. Experimental data suggests that the precise shape of the STDP windows is task dependent. A zoo of different functions has been identified, depending on the type of synapse [Caporale and Dan, 2008]. Most of these functions are monotonically decreasing with the magnitude of spike time difference. If the pre-synaptic spike arrives before the post-synaptic spike is emitted, the synaptic efficiency is increased and otherwise decreased, which reflects the causal relationship of interacting neurons.

4.2.3 Synaptic tagging

Besides the short lasting temporal dependence of pre- and post-synaptic spikes, more detailed studies of the biochemical mechanisms of LTP have revealed mechanisms on a longer time scale. Long-lasting synaptic strengthening involves gene expression in the neuron [Krug et al., 1984]. Since the synthesis of macromolecules occurs mainly in the soma the following question arises: How do synapses communicate with the long distant soma and how do the synthesised molecules find their way back to the right synapse? The *synaptic tag* hypothesis copes with this problem and is well supported experimentally [Frey and Morris, 1998]. It suggests that activated synapses are marked by setting up a synaptic tag. Macromolecules involved in LTP are synthesised at the soma and diffusely dispensed over the dendrites. Gene expression is triggered when the tags are set and LTP takes place when the molecules reach a tagged synapse. This hypothesis gives a relatively simple answer to the question above and also explains the paradoxical finding that LTP was also observed when gene expression was inhibited [Krug et al., 1984]. Since a reservoir of Macromolecules is persistent in the dendritic tree and synaptic tagging does not involve gene expression, LTP persists until the reservoir is used up [Frey and Morris, 1998].

Besides its functional necessity to implement LTP in neurons, synaptic tagging adds an extra dimension to Hebbian learning. The time window between setting the tags and arrival of macromolecules can be used to further modulate the change in synaptic efficiency. It has been shown, that during this critical time the strength of LTP can be modulated by reward. Dopamine is a neurotransmitter closely related to reward. If dopamine is blocked, the maintenance of LTP is suppressed [Frey et al., 1990], whereas dopamine release facilitates LTP [Otani et al., 2003]. These findings have inspired the work of [Izhikevich, 2007]. The authors proposed that reward modulated LTP could be the key to the distal reward problem. The distal reward problem arises form the fact that in a reinforcement learning task, the reward often arrives seconds or minutes after the cues that have led to the reward. Thus, the spike patterns the cues became manifest in, have already vanished when the reward arrives. [Izhikevich, 2007] proposed that this problem could be solved by using a reward modulated STDP rule. Synapses that took part in firing during the cue's spike patterns become tagged. When the reward arrives the tags are consolidated leading to LTP. On the other hand, tags that did not lead to reward transiently die out.

Other studies showed that the consolidation of synaptic tags also depends on the synaptic input activity. [Sajikumar and Frey, 2004] showed that low frequency stimulation (LFS) of hippocampal CA1 neurons, short after the induction of LTP, resets the synaptic tags. The authors suggested that LTP of a single synapse can be stopped in the early phase (5 minutes after induction) using LSF, preventing the formation of memory traces. This effect did not influence the efficiency of other synapses that already have consolidated LTP. In a related experiment it was shown that tag resetting is not achieved by suppressing gene expression, but seems to be a local synaptic process [Young and Nguyen, 2005]. Nevertheless, the authors found that the tag resetting mechanism affects LTP in the whole neuron. In particular, they found that the stability of the neuron's tags can be influenced by applying different input patterns to a single synapse. They reported that applying a short tetanic input to one synapse, prevented the resetting of tags at another synapse. Subsequent application of LFS did not influence the synaptic efficiency and led to normal LTP. This mechanism of active forgetting or active memory maintenance further extends the basic theory of Hebbian learning, for which we will derive a theoretical basis throughout this thesis.

4.3 Neural codes

Spikes are uniform events and thus fully described by their time of occurrence. Since spikes are used to exchange information, this implies that information is encoded in the brain by patterns of spike arrival times.

4.3.1 Spike and rate codes

A simple way to interpret the output of neurons is to assume that the information is solely captured in its average firing rate. The average is most commonly taken over a finite time window of the neuron's output. The length of the time window should contain several spikes to avoid strong fluctuations in the rate [Maass and Bishop, 2001]. This neural code is motivated by results that suggest that neural activity is largely influenced by noise processes. In fact, many experimental studies in vivo suggest that spike times are driven by a Poisson process [Softky and Koch, 1993]. It remains unclear whether this random behaviour is driven by noise injected with the input, or an intrinsic feature of the neuron. Many studies suggest that the former is the case. Features like correlation, synchrony and phase of spiking inputs have been shown to carry information that is used in neural processing [Gray and Singer, 1989, von der Malsburg and Buhmann, 1992, O'Keefe and Recce, 1993]. But in practice it can be shown that in many cases the results of spike and rate codes are identical if parameters are carefully chosen, e.g. by making the time window for averaging small enough [Gerstner and Kistler, 2002].

4.3.2 Winner-take-all circuits

Viewing the brain as a randomly connected set of neurons is not in accordance with the facts. Besides the large-scale anatomical areas, that differ in shape, size and functionality, also within neural microcircuits a lot of structure emerges. One of the best studied brain area in this regard is the neocortex, which has a well organised, layered structure [Douglas and Martin, 2004]. The layers 2 and 3 receive input from subcortical and other cortical areas. The output of these layers is then further processed in layer 5, which projects back to layers 2 and 3 and subcortical areas. Cells of the layers 2,3 and 5 are organised in patches that inhibit each other within a layer. This lateral inhibition gives rise to a competition between the neurons. Only neurons that were most excited by their input will fire a spike that suppresses activity of all other cells within the patch. If only the strongest neuron is selected these circuits are called winner-take-all (WTA), if neighbouring neurons are not completely suppressed, but strongly inhibited allowing them to fire with low but non-zero probability they are called soft WTA circuits. These circuits have been proven to be very powerful computational units. In particular [Maass, 2000] showed that a single layer of cells connected to a soft WTA circuit, with linear weights to an input layer has the universal approximator property.

WTA circuits provide another form of neural coding. Since each cell's firing represents the strongest response to an input pattern, while keeping all other cells of the patch silent, the identity of the active cell can be considered as a neural code. The information is encoded in the whole patch of neurons. If each cell reports its certainty about, being the winner neuron by its spike rate, the patch activity becomes a probabilistic population code, representing a probability distribution over all input identities [Zemel et al., 1998].

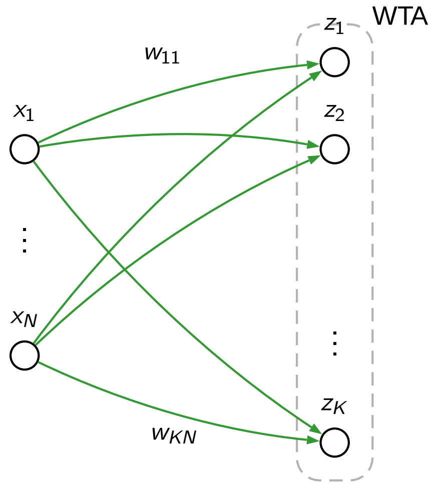


Figure 4.2: The SEM network structure. The network is composed of a single layer of neurons $z_1 \dots z_K$, that form a soft WTA circuit through lateral inhibition. The WTA layer receives feedforward connections from the inputs $x_1 \dots x_N$ with weights w_{ki}, indicated in green.

4.4 The spiking expectation-maximisation network

In section 4.3.2 we have outlined the computational power of soft WTA circuits. In this section we will review recent work that further benefits from their properties.

4.4.1 Bayesian inference in soft WTA circuits

The spiking expectation-maximisation (SEM) network was introduced recently [Nessler et al., 2010]. The authors proved that a soft WTA circuit can approximate a stochastic version of EM using a STDP rule. The SEM network is composed of a simple, feedforward, one-layer structure. The input is given by a vector $\mathbf{x} \in \{0,1\}^N$. It consists of N binary unites x_i that obey $\sum_i x_i = 1$. The output layer consists of K neurons that receive feedforward connections with weights w_{ki} between the i^{th} input and the k^{th} output, which are collected in a matrix $\mathbf{W}$. In addition, each output neuron has a bias weight w_{k0}, collected in a vector $\mathbf{w_0}$. See figure 4.2 for a schematic view of the network structure.

The membrane potential of neuron k is computed by a linear combination of the weighted inputs

$$u_k = \sum_{i=1}^{N} w_{ki} x_i + w_{k0}. \tag{4.7}$$

The neurons produce a binary output vector $\mathbf{z} \in \{0, 1\}^K$. It is assumed that the output neurons inhibit each other, to form a soft WTA network. On that account, the probability that the k^{th} unit produces a spike, is given by the *soft-max* function:

$$p\left(z_k \mid \mathbf{x}, \mathbf{w_0}, \mathbf{W}\right) = \frac{\mathrm{e}^{u_k}}{\sum_{l=1}^{K} \mathrm{e}^{u_l}}. \tag{4.8}$$

It was shown, that if the weights are equal to the target values

$$w^*_{ki} = \log p\left(x_i \mid z_k\right), \qquad w^*_{k0} = \log p\left(z_k\right), \tag{4.9}$$

the network computes Bayesian inference. Thus, the output probabilities are equal to posterior distributions of the hidden causes for the input stimulus. The output spikes generated by the network can be interpreted as drawing samples form this posterior distribution.

4.4.2 A STDP rule for EM

In the previous section we have argued that, if the network weights were given by the target weights (4.9) the network would compute Bayesian inference. It was further shown that the target weights can be reached using a STDP rule. Due to the logarithmic form of the weights an update rule can be found, which was reported in [Nessler et al., 2010] to be

$$\Delta w_{ki} = \begin{cases} \eta\left(\mathrm{e}^{-w_{ki}}\, x_i - 1\right), & \text{if } z_k = 1 \\ 0, & \text{if } z_k = 0 \end{cases}, \qquad \Delta w_{k0} = \eta\left(\mathrm{e}^{-w_{k0}}\, z_k - 1\right). \tag{4.10}$$

This is a purely local rule, that depends only on the current value of the weights and the correlation between pre- and post-synaptic spikes. In [Nessler et al., 2010] several STDP windows were reported that approximate the rectangular shape of the rule. The target weights (4.9) are the only equilibrium points of the optimal rule (4.10) and they are reached exponentially fast.

It can be shown that the application of the learn rule (4.10), is equal to the application of stochastic EM. Each output spike generated by the network can be interpreted as a stochastic E step, the weight update was proven to move always into the direction of the M step. Thus, the network proposed is able to learn a model of the form shown in figure 3.1, where the distribution of $\mathbf{x}$ is given by a mixture of multinomials [Nessler et al., 2010]. The model can be trained in an online fashion and fully unsupervised, by presenting a series of input patterns. The networks can also be used to build hierarchies. Due to the softmax function (4.8) the output obeys the same constraint as the input $\sum_k z_k = 1$, multiple SEM networks can be stacked onto each other to form deep multi-layered structures.

[Nessler et al., 2010] experimentally verified the computational power of the network by training it to distinguish between handwritten digits. The digits were presented through 858 spiking neurons, each of which corresponded to one pixel of the digit images. The inputs were binary, a Poisson rate of 40 Hz corresponded to a black pixel, 0 Hz to a white pixel. A Poisson process caused firing of the WTA circuit approximately every 5ms. Ten neurons were trained, three of which adapted to one of the three presented digit identities. The network learned fully unsupervised to distinguish between the three digit classes.

4.4.3 Relation to the LIF neuron

A biologically more realistic implementation of the SEM network was established in [Nessler et al., 2010]. They used a double exponential EPSP kernel, as introduced in section 4.1.2 with zero delays on the dendrites to learn spatiotemporal patterns. The patterns were generated by drawing fixed spike times from a Poisson distribution with 15Hz for each of the 500 input channels. They created patterns of 50ms length and presented them to the SEM network alternating with frames of pure spike noise. The network quickly adapted to the input, representing the pattern identity in the output activity and showed robustness to noise and time warping.

In [Habenschuss, 2010] a generalisation of the model was introduced. Similar learn rules to that given in equation (4.10), can be used to learn mixtures of any probability distribution from the exponential family, when the network inputs are given by their sufficient statistics. Of particular interest in the context of neuron modelling is the Poisson distribution. The input variables x_i have to be equal to the rates of the i^{th} input, which can be achieved by using a rectangular EPSP kernel, averaging the input spikes over a window of lenght T

$$\alpha(t) = \begin{cases} 1 & \text{if } 0 \leq t \leq T \\ 0 & \text{else} \end{cases} . \tag{4.11}$$

The membrane potential was calculated by a slight modification of equation (4.7). Using this representation of the input, equation (4.10) was proven to learn a mixture of Poisson distributions.

Processing Poisson distributions allows the network to deal with spiking inputs. A spiking output can be generated by letting the whole network fire with an overall Poisson rate λ. The k^{th} neuron's firing rate is then given by

$$\lambda_k^{(t)} = \lambda \cdot \frac{\mathrm{e}^{u_k^{(t)}}}{\sum_{l=1}^{K} \mathrm{e}^{u_l^{(t)}}}, \tag{4.12}$$

Each neuron fires with its individual spike rate $\lambda_k^{(t)}$. Thus, in a discrete time simulation it is possible, that multiple neurons emit a spike simultaneously. Although this case is not supported by theory, we will show experimentally that this simple spiking neuron model achieves good learning results.

4.5 Neural implementations of HMMs

The relation between HMMs and neural networks has been proposed by several authors. [Bobrowski et al., 2009] have shown that recurrent neural networks can implement Bayesian inference for HMMs. The transition probabilities of the HMM were implemented by lateral connections between the neurons, the observation probabilities by feed forward connections from the input. Using a specific parametrisation for the weights, a spiking network was able to implement Bayesian filtering, but the authors did not provide a learn rule for these weights.

In an approach similar to the SEM network a spiking neural model was found that is capable of doing Bayesian inference on a binary hidden cause variable [Deneve, 2008a]. The model was named *Bayesian spiking neuron*. The neurons are at any time in one of two states, the *on* or the *off* state. They keep track of the log-odds ratio of the hidden states. The underlying model is assumed to be a two-state hidden Markov model over the history of input spikes. In addition to the true log-odds ratio, that is computed on the neuron's inputs, the model keeps track of a predicted value of the log-odds ratio computed on the output. If the observed input exceeds the internal representation a spike is emitted. On this account the neuron reports only new information, that is not captured by the internal model. This results in a very efficient neural code, that uses the minimum number of spikes. This neural model is very similar to the leaky integrate-and-fire neuron. The true log-odds can be interpreted as the neuron's membrane potential, the predicted log-odds as a time-varying threshold for spike generation. The spike generation is deterministic, but the output spike distribution has been shown experimentally to be close to a Poisson distribution, if the inputs are Poisson distributed. Since input and output follow approximately the same distribution, the network can be used to form hierarchies.

In the companion letter [Deneve, 2008b] the parameter learning for the Bayesian spiking neuron was derived. It was shown that the cells can learn the HMM parameters using a STDP rule. The computations are similar to the online expectation-maximisation algorithm that was introduced in [Mongillo and Deneve, 2008] and thus, the neuron has to keep track of the sufficient statistics of the input. To achieve this, learning is not performed just on the firing times of a single input and output spike, but on a sliding window over the neural inputs including future and past spike events. The exact neural mechanisms for computing the sufficient statistics were not reported by the authors. Extending this model to HMMs with multiple hidden states is not straightforward, since both computations for inference and learning benefit form simplifications that are possible in a two state HMM only. Distributing the hidden cause information over a network of multiple binary neurons is also not trivial, since this would involve sharing information of the neuron's internal states over the whole network. To the best of our knowledge the Bayesian spiking neuron is the only neural model that can learn HMMs using a STDP-like rule.

5 STDP learning of temporal hidden causes

In the previous chapter we have reviewed the recent results introduced in [Nessler et al., 2010]. In this chapter we will extend this idea to enable the network to detect relationships between temporal hidden causes from an input spike stream. We will investigate two approaches, both are based on extending the input of the network:

- In section 5.1 we will extend the network inputs with time lags. We will see that this simple extension significantly increases the computational power of the network.
- In section 5.2 we will extend the network with lateral excitatory connections. We will show that this network is able to perform a Monte Carlo approximation of HMM inference and learning.

5.1 Discovering temporal hidden causes using multiple time-lags

In the original work on SEM networks the spike trains of all inputs arrived at the soma simultaneously [Nessler et al., 2010]. In a biological neuron however, different synapses on the dendritic tree of a single cell have different delays. These delays arise from the distances to the soma. Synapses that are far away have larger delays due to finite propagation speed of the EPSPs. A more realistic mathematical representation of the input activation over the dendrites was given by the EPSP kernel in equation (4.5). Each synapse i contributes a specific delay Δ_i to the input spikes. We will use this input representation here to capture temporal relations between nearby time windows of the input.

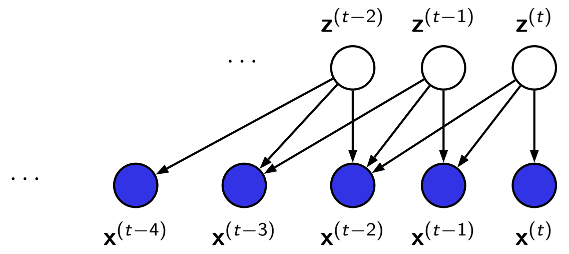


Figure 5.1: The Bayesian network of a time lagged SEM model. Here the input is presented on three time lags, $\Delta_1 = 0$, $\Delta_2 = 1$ and $\Delta_3 = 2$. Each hidden cause depends on the most recent input history.

The network input is extended by presenting it to the network at M different time lags Δ_m, $m = 1 \dots M$. Thus, the network input vector is given by

$$\mathbf{y}^{(t)} = \left[\mathbf{x}^{(t-\Delta_1)}, \mathbf{x}^{(t-\Delta_2)}, ..., \mathbf{x}^{(t-\Delta_M)}\right]. \tag{5.1}$$

$\mathbf{y}^{(t)}$ is a vector of size $N \cdot M$, where N is the size of $\mathbf{x}^{(t)}$. The network benefits from this extension by being able to extract statistical information from the input at multiple points in time simultaneously (figure 5.1 shows an example). The hidden causes depend on all time-lagged inputs and thus capture temporal correlations. The membrane potential is calculated according to (4.8)

$$u_k^{(t)} = \sum_{i=1}^{N} w_{ki} y_i^{(t)} + w_{k0}. \tag{5.2}$$

Using the SEM learn rule (4.10), the synaptic weights converge to the log probabilities

$$w_{ki}^* = \log p\left(y_i \mid z_k\right), \qquad w_{k0}^* = \log p\left(z_k\right). \tag{5.3}$$

We find for the network output probability, with respect to these target weights

$$\begin{aligned} p\left(z_k^{(t)} \mid \mathbf{x}^{(t-\Delta_1)}, \dots, \mathbf{x}^{(t-\Delta_M)}\right) &= \frac{\mathrm{e}^{u_k}}{\sum_{l=1}^{K} \mathrm{e}^{u_l}} \\ &= \frac{p\left(z_k^{(t)}\right) \prod_{m=1}^{M} p\left(\mathbf{x}^{(\tau-\Delta_m)} \mid z_k^{(t)}\right)}{\sum_{l=1}^{K} p\left(z_l^{(t)}\right) \prod_{m=1}^{M} p\left(\mathbf{x}^{(\tau-\Delta_m)} \mid z_l^{(t)}\right)}. \end{aligned} \tag{5.4}$$

The output probabilities are equivalent to the inference of the hidden cause $z_k^{(t)}$ given the time lagged inputs, which can be seen by inspecting the Bayesian network from figure 5.1.

The hidden cause's dependence on multiple time lags introduces a serious challange to the model. Consecutive hidden causes in the Bayesian network from figure 5.1 share child nodes, which entails the problem of *explaining away* [Bishop, 2006]. The hidden causes can no longer be assumed to be independent for a given input sequence $\mathbf{X}$. To see this, we write down the conditional probability of a hidden cause sequence $\mathbf{Z}$ given $\mathbf{X}$

$$\begin{aligned} p(\mathbf{Z} \mid \mathbf{X}) &= \frac{p(\mathbf{Z}, \mathbf{X})}{\sum_{\mathbf{Z}'} p(\mathbf{Z}', \mathbf{X})} \\ &= \frac{\prod_{\tau=1}^{T} p\left(\mathbf{z}^{(\tau)}\right) \prod_{m=1}^{M} p\left(\mathbf{x}^{(\tau-\Delta_m)} \mid \mathbf{z}^{(\tau)}\right)}{\sum_{\mathbf{Z}'} \prod_{\tau=1}^{T} p\left(\mathbf{z}'^{(\tau)}\right) \prod_{m=1}^{M} p\left(\mathbf{x}^{(\tau-\Delta_m)} \mid \mathbf{z}'^{(\tau)}\right)}. \end{aligned} \tag{5.5}$$

Since the denominator is a sum of products over $\mathbf{Z}$, equation (5.5) will in general not factorise into a product of terms depending on single hidden causes $\mathbf{z}^{(\tau)}$. Thus, we find that the hidden causes are no longer independent. However, the network samples a path $\mathbf{Z}$ by drawing an individual sample from equation (5.4) in each time step. The whole-path probability is given by the product of all individual sampling probabilities

$$\begin{aligned} q(\mathbf{Z} \mid \mathbf{X}) &= \prod_{\tau=1}^{T} \frac{p\left(\mathbf{z}^{(\tau)}\right) \prod_{m=1}^{M} p\left(\mathbf{x}^{(\tau-\Delta_m)} \mid \mathbf{z}^{(\tau)}\right)}{\sum_{\mathbf{Z}'} p\left(\mathbf{z}'^{(\tau)}\right) \prod_{m=1}^{M} p\left(\mathbf{x}^{(\tau-\Delta_m)} \mid \mathbf{z}'^{(\tau)}\right)} \\ &= \frac{p(\mathbf{Z} \mid \mathbf{X})}{\prod_{\tau=1}^{T} \sum_{\mathbf{Z}'} p\left(\mathbf{z}'^{(\tau)}\right) \prod_{m=1}^{M} p\left(\mathbf{x}^{(\tau-\Delta_m)} \mid \mathbf{z}'^{(\tau)}\right)}. \end{aligned} \tag{5.6}$$

Comparing equation (5.6) with the true conditional (5.5), we find that the network implicitly introduced an independence assumption between the hidden cause variables, since equation (5.6) is factorisable over the hidden causes. This simplification introduces a bias to the learning. Although we will see in chapter 6 that using this biased learn rule the network can achieve good performance in solving some practical problems, it will lead to suboptimal results in general. In the next section we will derive an extended learn rule for HMMs that copes with a similar problem. We will find that this extension is also capable to correct the bias of the model introduced above.

5.2 STDP learning of HMMs

In section 4.4 we have argued that a winner-take-all circuit can approximate a stochastic version of EM using a simple STDP rule. We will extend this

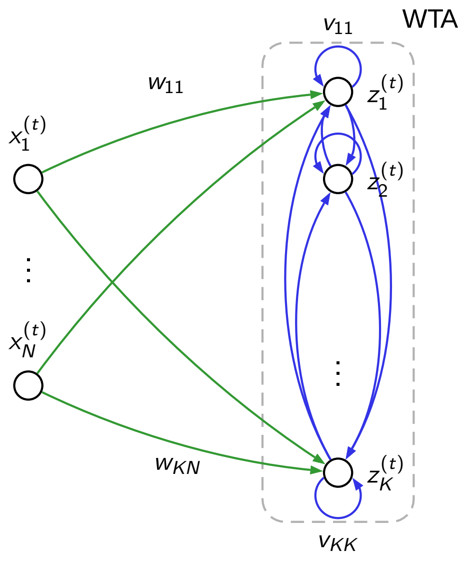


Figure 5.2: A SEM circuit with recursive connections to capture the temporal structure of the input.

approach here to capture the temporal structure of the input by introducing lateral connections. We will show that this network structure can do inference on the HMM's hidden states and Monte Carlo EM can be used to learn the network weights.

5.2.1 Introducing temporal relations between hidden causes through lateral connections

The network structure of the extended SEM circuit is shown in figure 5.2. Like in the original SEM network, the input layer consists of N binary unites $x_i^{(t)}$ that represents a population code of a time varying input variable. The inputs are collected in a vector $\mathbf{x}^{(t)} \in \{0,1\}^N$. The output layer consists of K neurons that receive feed-forward connections with weights w_{ki} from the input and lateral connections v_{kj} between the output layer units. The feed-forward and lateral weights are collected in the matrices $\mathbf{W}$ and $\mathbf{V}$ respectively. The neurons produce a binary output vector $\mathbf{z}^{(t)} \in \{0,1\}^K$ in each time step. Through the lateral connections the output of the previous time step $\mathbf{z}^{(t-1)}$ is fed back to the network. In this setup, the layer of WTA neurons is a recurrent neural network with all-to-all connectivity. As in the original SEM network, the probability that unit k fires at time t, is given by the *soft-max* function:

$$\begin{aligned} p\left(z_k^{(t)} \mid \mathbf{x}^{(t)}, \mathbf{z}^{(t-1)}, \mathbf{W}, \mathbf{V}\right) &= \frac{\mathrm{e}^{u_k^{(t)}}}{\sum_{l=1}^{K} \mathrm{e}^{u_l^{(t)}}}, \\ \text{with } u_k^{(t)} &= \sum_{i=1}^{N} w_{ki} x_i^{(t)} + \sum_{j=1}^{K} v_{kj} z_j^{(t-1)}, \end{aligned} \tag{5.7}$$

where $u_k^{(t)}$ is the *membrane potential* of cell k at time t. The soft-max representation avoids zero probabilities, which is in general a desirable property for HMMs [Baldi and Chauvin, 1994].

At this point we notice that, if the weights of the network were given by

$$\begin{aligned} w_{ki}^{*} &= \log p\left(x_i^{(t)} \mid z_k^{(t)}\right), \\ v_{kj}^{*} &= \log p\left(z_k^{(t)} \mid z_j^{(t-1)}\right), \end{aligned} \tag{5.8}$$

the output probability (5.7) would become equivalent to the inference of the state at time t in a HMM, given the previous state and current observation. By substituting equation (5.8) into (5.7) we get

$$\begin{aligned} p\left(z_k^{(t)} \mid \mathbf{x}^{(t)}, \mathbf{z}^{(t-1)}\right) &= \frac{\mathrm{e}^{u_k^{(t)}}}{\sum_{l=1}^{K} \mathrm{e}^{u_l^{(t)}}} \\ &= \frac{p\left(\mathbf{x}^{(t)} \mid z_k^{(t)}\right) p\left(z_k^{(t)} \mid \mathbf{z}^{(t-1)}\right)}{\sum_{l=1}^{K} p\left(\mathbf{x}^{(t)} \mid z_l^{(t)}\right) p\left(z_l^{(t)} \mid \mathbf{z}^{(t-1)}\right)}, \end{aligned} \tag{5.9}$$

where we have dropped the direct dependence on the network weights assumed to be constant. Thus, the output layer of the network represents in every time step, the hidden state of a HMM inferred from the previous state and the current observation. In the remainder of this discussion we will show how the target weights (5.8) can be learned.

5.2.2 A sampling approximation of the forward-backward algorithm

In section 3.5.1 we have seen that although the forward-backward algorithm can be computed efficiently when the observation is given as a whole batch of length T, the calculation of the backward messages becomes very difficult when the algorithm is applied on incrementally arriving data. In the neural implementation of HMMs, using the recurrent architecture shown in figure 5.2 we can no longer assume an input sequence of length T to be given. The current observation will only be transiently available by the spike patterns arriving at the neuron's inputs. Applying standard forward-backward equations in such an environment is not feasible, as we have argued above. In this section we will introduce an approximate algorithm that uses Monte Carlo techniques to solve this problem. In our discussion we will focus on algorithms that can be implemented in an online setup, it should be stressed however that these algorithms would also work in a batch application.

We assume, that at the beginning of a sequence one neuron z_0 is active, that projects to neuron k of the WTA population with a synaptic strength given by

$$v_{k0}^{*} = \log p\left(z_k^{(1)}\right), \tag{5.10}$$

which models the HMM initial state probabilities. To simplify the notation let us use the convention $p\left(\mathbf{z}^{(1)} \mid \mathbf{z}^{(0)}\right) \equiv p\left(\mathbf{z}^{(1)}\right)$. Then, the probability of sampling a path $\mathbf{Z}$ using equation (5.9) is given by

$$\begin{aligned} q\left(\mathbf{Z} \mid \mathbf{X}\right) &= \prod_{\tau=1}^{T} p\left(\mathbf{z}^{(\tau)} \mid \mathbf{x}^{(\tau)}, \mathbf{z}^{(\tau-1)}\right) \\ &= \prod_{\tau=1}^{T} \frac{p\left(\mathbf{x}^{(\tau)} \mid \mathbf{z}^{(\tau)}\right) p\left(\mathbf{z}^{(\tau)} \mid \mathbf{z}^{(\tau-1)}\right)}{\sum_{l=1}^{K} p\left(\mathbf{x}^{(\tau)} \mid z_l^{(\tau)}\right) p\left(z_l^{(\tau)} \mid \mathbf{z}^{(\tau-1)}\right)} \\ &= \frac{p\left(\mathbf{Z}, \mathbf{X}\right)}{\prod_{\tau=1}^{T} \sum_{l=1}^{K} p\left(\mathbf{x}^{(\tau)} \mid z_l^{(\tau)}\right) p\left(z_l^{(\tau)} \mid \mathbf{z}^{(\tau-1)}\right)}. \end{aligned} \tag{5.11}$$

The numerator is the correct HMM joint distribution (3.11). The denominator, however is not the accurate normalisation of the conditional distribution and thus we will find in general that $q\left(\mathbf{Z} \mid \mathbf{X}\right) \neq p\left(\mathbf{Z} \mid \mathbf{X}\right)$.

Before we continue let us explore some properties of the distribution (5.11). We rewrite the transition probability between two states at time t for a given observation (3.12) in terms of the distribution (5.11)

$$\xi_q\left(\mathbf{z}^{(t)}, \mathbf{z}^{(t-1)}\right) = \sum_{\mathbf{Z} \setminus \mathbf{z}^{(t)}, \mathbf{z}^{(t-1)}} q\left(\mathbf{Z} \mid \mathbf{X}\right)$$

$$
\begin{aligned}
&= \sum_{\mathbf{Z}\backslash\mathbf{z}^{(t)},\mathbf{z}^{(t-1)}} \prod_{\tau=1}^{T} p\left(\mathbf{z}^{(\tau)} \mid \mathbf{x}^{(\tau)}, \mathbf{z}^{(\tau-1)}\right) \\
&= \sum_{\mathbf{z}^{(1)}\ldots\mathbf{z}^{(t-2)}} \prod_{\tau=1}^{t} p\left(\mathbf{z}^{(\tau)} \mid \mathbf{x}^{(\tau)}, \mathbf{z}^{(\tau-1)}\right) \times \\
&\quad \left(\sum_{\mathbf{z}^{(t+1)}\ldots\mathbf{z}^{(T)}} \prod_{\tau=t+1}^{T} p\left(\mathbf{z}^{(\tau)} \mid \mathbf{x}^{(\tau)}, \mathbf{z}^{(\tau-1)}\right) \right)
\end{aligned}
$$

The second term sums to one, thus we can write

$$
\begin{aligned}
\xi_q\left(\mathbf{z}^{(t)}, \mathbf{z}^{(t-1)}\right) &= \sum_{\mathbf{z}^{(1)}\ldots\mathbf{z}^{(t-2)}} \prod_{\tau=1}^{t} p\left(\mathbf{z}^{(\tau)} \mid \mathbf{x}^{(\tau)}, \mathbf{z}^{(\tau-1)}\right) \\
&= p\left(\mathbf{z}^{(t)}, \mathbf{z}^{(t-1)} \mid \mathbf{X}^{(1\ldots t)}\right). \qquad (5.12)
\end{aligned}
$$

If we compare this result to the standard forward-backward equations (3.17), we see that (5.12) computes the forward messages correctly but does not take the backward messages into account. Unlike the hidden Markov model would suggest, all transitions are independent of future observations, which introduces a bias.

Next we observe that, since the denominator is always between zero and one, equation (5.11) will be zero only where $p(\mathbf{Z}, \mathbf{X})$ is zero[1]. It holds true that

$$
q(\mathbf{Z} \mid \mathbf{X}) \neq 0 \quad \Leftrightarrow \quad p(\mathbf{Z}, \mathbf{X}) \neq 0. \qquad (5.13)
$$

Hence equation (5.11) can be used in the framework of importance sampling as a proposal distribution for $p(\mathbf{Z}, \mathbf{X})$. The bias introduced by ignoring the backward messages can be corrected by using importance weights, given by

$$
r(\mathbf{Z}) = \frac{p(\mathbf{Z}, \mathbf{X})}{q(\mathbf{Z} \mid \mathbf{X})} = \prod_{\tau=1}^{T} \sum_{l=1}^{K} p\left(\mathbf{x}^{(\tau)} \mid z_l^{(\tau)}\right) p\left(z_l^{(\tau)} \mid \mathbf{z}^{(\tau-1)}\right),
$$

which can be updated recursively

$$
r(\mathbf{Z}) = r\left(\mathbf{Z}^{1\ldots T-1}\right) \times \sum_{l=1}^{K} p\left(\mathbf{x}^{(T)} \mid z_l^{(T)}\right) p\left(z_l^{(T)} \mid \mathbf{z}^{(T-1)}\right). \qquad (5.14)
$$

[1]The pathological case where the denominator of equation (5.11) becomes zero is implicitly avoided by the use of the logarithmic parametrisation of the proposed network.

Paths sampled from the proposal distribution are generated by the network architecture we have presented in the previous section, by repeatedly evaluating equation (5.9). We will refer to this path generation as *forward sampling*. To correct the bias, the sampled paths must be accumulated in a weighted sum similar to equation (3.7). We assume that a set of paths $\mathbf{Z}_l$, $l \in \{1 \dots L\}$, was generated for a given observation $\mathbf{X}$ and the importance weights were calculated using (5.14). To approximate the E step equation (3.12) we would like to find an expression for the conditional distribution $p\left(\mathbf{Z} \mid \mathbf{X}\right)$ rather than the joint $p\left(\mathbf{Z}, \mathbf{X}\right)$. Note however that since the proposal is already conditioned on $\mathbf{X}$ a re-normalisation can be easily achieved by forcing all importance weights to sum to one. Using this we can write down the approximation of equation (3.12) given by

$$\begin{aligned} \gamma\left(z_k^{(t)}\right) &\approx \frac{1}{\sum_{m=1}^{L} r\left(\mathbf{Z}_m\right)} \sum_{l=1}^{L} r\left(\mathbf{Z}_l\right) \cdot z_{k,l}^{(t)} \\ \xi\left(z_k^{(t)}, z_j^{(t-1)}\right) &\approx \frac{1}{\sum_{m=1}^{L} r\left(\mathbf{Z}_m\right)} \sum_{l=1}^{L} r\left(\mathbf{Z}_l\right) \cdot z_{k,l}^{(t)} \cdot z_{j,l}^{(t-1)}, \end{aligned} \tag{5.15}$$

where $z_{k,l}^{(t)}$ is the k^{th} element of the sampled path $\mathbf{Z}_l$ at time t. The algorithm can be summarised as follows:

1. For a given observation $\mathbf{X}$, sample L paths from the model using equation (5.9) and the current model parameters.
2. Calculate the importance weights for each sampled path using equation (5.14).
3. Calculate the approximate E Step using equation (5.15).
4. Update the model parameter using the update equations (3.13).

This algorithm converges to the Baum-Welch results for $L \to \infty$. It works in both an online and a batch setup. If the algorithm is used in the batch mode, the paths can be sampled one after another and together with the weights, they can be accumulated in a matrix. When all paths have been sampled the update rules can be applied on the whole batch using the exact M step. In the online setup, L paths are sampled for a given sequence, the importance weights can be calculated incrementally. For this approach to make sense also an online version of the M step must be found. This problem will be addressed in the next section, where we will show that a STDP rule with synaptic tagging can be used to solve this task.

5.3 A STDP Rule for HMM Learning

We will derive the learning rule for the recurrent weights v_{kj} only, since adaptation for the feed-forward and initial state weights is straightforward. Let us introduce the weight update rule for the recurrent state connections, which is a slight modification of the STDP rule that was introduced in [Nessler et al., 2010]

$$\Delta v_{kj}^{(t)} = \begin{cases} \eta\,(\mathrm{e}^{-v_{kj}} - 1)\,, & \text{if } z_k^{(t)} = 1 \text{ and } z_j^{(t-1)} = 1 \\ -\eta, & \text{if } z_k^{(t)} = 0 \text{ and } z_j^{(t-1)} = 1\,, \\ 0, & \text{if } z_j^{(t-1)} = 0 \end{cases} \tag{5.16}$$

where η is the learn rate. For the HMM learning task we derive the equilibrium point analysis for a sequence of length T. The application of the learn rule will be governed by n_{kj}^* the expected numbers of $z_j \rightarrow z_k$ transitions and n_j^* the expected number of times spent in state j, over the whole sequence. Using equivalent calculations as in [Nessler et al., 2010] we get for the equilibrium point of equation (5.16)

$$\begin{aligned} \mathbb{E}\{\Delta v_{kj}^{(t)}\} = 0 \quad &\Leftrightarrow \quad \eta\left(\mathrm{e}^{-v_{kj}} - 1\right) n_{kj}^* - \eta \sum_{l \neq k} n_{lj}^* = 0 \\ &\Leftrightarrow \quad \mathrm{e}^{-v_{kj}}\, n_{kj}^* = n_j^* \\ &\Leftrightarrow \quad v_{kj} = \log \frac{n_{kj}^*}{n_j^*}, \end{aligned}$$

Note that if n_{kj}^* and n_j^* were equal to the quantities given in equation (3.14) the weights updated by rule (5.16) would converge to the correct target distribution (5.8). Unfortunately the network samples from the proposal distribution $q\,(\mathbf{Z} \mid \mathbf{X})$ and thus the transition and state probabilities are biased. To correct this bias we must implement the importance weights into the STDP rule.

In order to adapt the STDP rule to the importance sampling let us first introduce an inhibitory neuron that creates the sum over all output activations and inhibits each neuron of the population with a current

$$i^{(t)} = \sum_{l=1}^{K} \mathrm{e}^{u_l^{(t)}}\,. \tag{5.17}$$

Figure 5.3 illustrates the inhibitory neuron. Note that $i^{(t)}$ is equivalent to the denominator of the soft-max function. To this end, it can be used by each

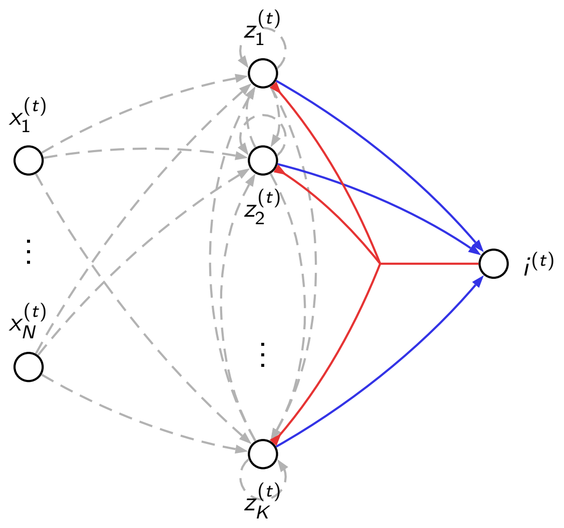


Figure 5.3: Extending the network with an inhibitory neuron to calculate the importance weights.

neuron to compute equation (5.7) locally. Further note that the inhibition is equivalent to the update term in equation (5.14), thus it can also be used to calculate the importance weights, which can be updated online using equation (5.17)

$$r\left(\mathbf{Z}\right) = r\left(\mathbf{Z}^{1\ldots T-1}\right) \times i^{(t)}. \tag{5.18}$$

The update rule can not be applied instantaneously because the importance weights need to be accumulated over a time window of length T. In the neural implementation we can achieve this by a synaptic tagging mechanism. This approach is similar to that in [Izhikevich, 2007] but here it is applied to an unsupervised learning task. Instead of updating the weights directly they get tagged and consolidation of the tags is delayed until the whole sequence was read. The strength of the tags is initialised to the weight changes (5.16) and then modulated by reward signals which are equivalent to the importance weights. The weight update over a whole sequence is given by

$$v_{kj}^{new} = v_{kj} + r\left(\mathbf{Z}\right) \times \sum_{\tau=1}^{T} \Delta v_{kj}^{(\tau)}. \tag{5.19}$$

Again several paths need to be sampled to get an accurate approximation of the statistics. The weight update for a set of L sampled paths can be written

(a) π_k

k	
1	.5
2	.0
3	.0
4	.5
5	.0
6	.0

(b) a_{kj}

$j\backslash k$	1	2	3	4	5	6
1	.0	.9	.0	.0	.0	.0
2	.0	.0	.9	.0	.0	.0
3	.0	.0	.9	.0	.0	.0
4	.0	.0	.0	.0	.9	.0
5	.0	.0	.0	.0	.0	.9
6	.0	.0	.0	.0	.0	.9

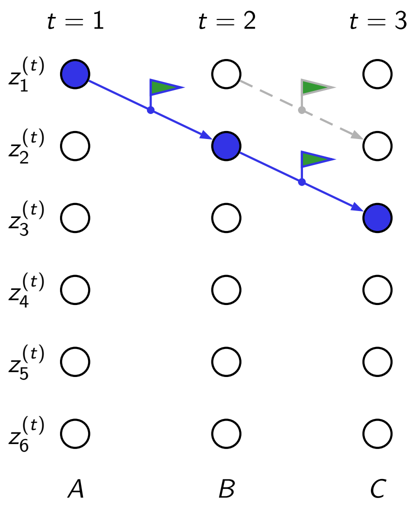
(c) b_{ki}

$k\backslash \mathbf{x}^{(t)}$	A	B	C	D
1	.9	.0	.0	.0
2	.0	.9	.0	.0
3	.0	.0	.9	.0
4	.9	.0	.0	.0
5	.0	.9	.0	.0
6	.0	.0	.0	.9

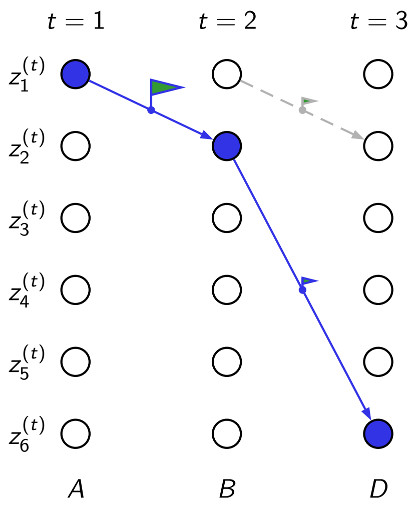
Table 5.1: The model parameter of the example HMM. (a) the initial state prior, (b) the transition probabilities, (c) the observation probabilities. Values are rounded to one digit precision.

$$v_{kj}^{new} = v_{kj} + \sum_{l=1}^{L} r\left(\mathbf{Z}_l\right) \times \sum_{\tau=1}^{T} \Delta v_{kj,l}^{(\tau)}. \tag{5.20}$$

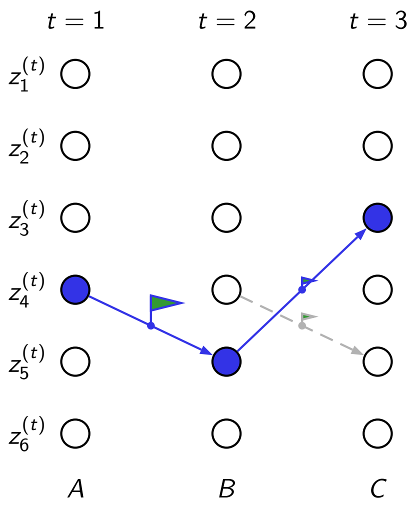
We are still missing the normalisation over all drawn paths to ensure the path probabilities to be normalised given the observation $\mathbf{X}$. Unfortunately unlike in the Baum-Welch equations this normalisation does not cancel out in our STDP rule but remains in the results as a bias. Introducing this normalisation into the neural implementation is not trivial. The easiest way to implement it would be to have L networks sample a path independently and in parallel. The importance weights are then normalised over all networks. This setup would be an efficient implementation of the importance sampling but it remains unclear how it could be implemented in a biologically plausible way since all networks would have to share the same weights and some mechanism needs to be introduced that keeps all importance weights normalised. A second approach is more biologically plausible: The model is represented in a single network and paths are sampled one after another. It is assumed that an input $\mathbf{X}$ is held in a short term memory such that it can be repeatedly presented. The bias is corrected by replaying the same input multiple times. The number of times the input is repeated is proportional to the importance weight. We will discuss this approaches later in more detail, let us first turn to a small toy example that illustrates the weight tagging mechanism.



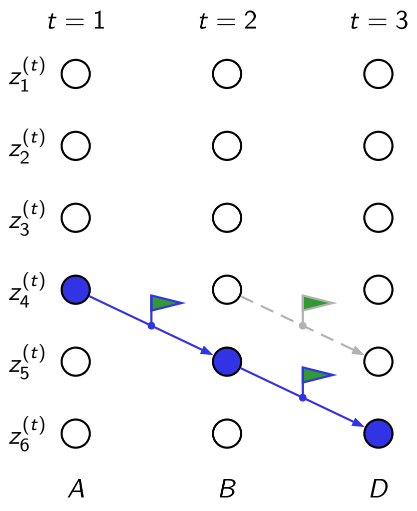
(a) Accepting a path for A-B-C



(b) Rejecting a path for A-B-D



(c) Rejecting a path for A-B-C



(d) Accepting a path for A-B-D

Figure 5.4: Illustration of the weight tagging. The sampling is shown unfolded over time. At each time step the active neuron is indicated by blue color and the weight tags by green flags. See text for details.

5.3.1 An illustrative example

To illustrate the weight tagging consider a signal source that produces sequences of four symbols A, B, C and D. The sequences are produced by successively generating words A-B-C or A-B-D. Each of the two words is produced with 50% probability. A HMM that can model this signal source is give by the model parameters in table 5.1. The model has 6 states: z_1 and z_4 encode symbol A, z_2 and z_5 encode symbol B, z_3 encodes symbol C and z_6 symbol D. Each of the two words is encoded by a unique state sequence $\{z_1, z_2, z_3\}$ for A-B-C and $\{z_4, z_5, z_6\}$ for A-B-D. The transition probabilities within each of the state sequences are high and all that are not part of a sequence are close to zero. Since one of the two words is generated randomly the initial probabilities of z_1 and z_4 are 0.5 and zero for all others. States z_3 and z_6 are terminal and thus self-recurrent. Note that this HMM is not the best choice to model this simple task, but we shall use it here to demonstrate the weight tagging.

The HMM from table 5.1 has one big disadvantage: it is ambiguous. For an observation sequence A-B both state sequences $\{z_1, z_2\}$ and $\{z_4, z_5\}$ are possible. Only on receiving the third symbol C or D one of the two state sequences becomes more likely. In the Baum-Welch algorithm this ambiguity is resolved by taking the backward messages into account which allow us to adapt the path probabilities to future events. Since this is not possible with our sampling approach, the best thing we can do is to guess and decide on one of the state sequences. As the crucial third symbol arrives we gain information whether our choice was good or not and we require this information to influence the weight update of the whole sampled path. This is achieved by the weight tagging, which is illustrated in figure 5.4. The figure shows the sampling unfolded over time, such that the state of the network in each time step is presented in each column. The four possible combinations of drawn states and inputs are shown. In each time step a winner neuron is drawn using equation (5.7). To keep the figure uncluttered only the weights that are currently updated are indicated by arrows linking two neurons. Each weight that lies between two successively drawn cells becomes tagged, indicated by small flags. The size of the flags indicates the current importance weight for the sequence, which is updated according to the current inhibition at each time step. Tags that have been set in previous time steps remain active for the whole sequence, indicated in gray.

Figures 5.4(a) and 5.4(c) show two sampled paths for the observed word A-B-C, 5.4(d) and 5.4(b) for A-B-D. The first path starts for A-B-C in state z_1, the second in state z_4 for the observation A. The next observation is B for which the most probable state transitions, according to table 5.1, are $z_1 \to z_2$ and $z_4 \to z_5$. In each time step the importance weights are indicted by the size of the flags. All markers that are currently present are displayed in gray the new arriving marker is shown in blue. Since the transitions in the first two time steps had high probabilities the inhibition is close to one and thus the markers keep their initial height. Next a symbol C arrives. The only state in which this symbol can be observed is z_3. For the fist path this state can be reached with high probability, for the second however a transition to state z_3 is very unlikely. This causes the inhibition to drop close to zero, which does not only effect the current weight update, but also tags that were set up previously. The whole path that was generated so far is devalued. For the second word A-B-D we observe a similar behaviour, but this time the first path turns out to be in the worse sequence as symbol D arrives, which causes the importance weight to be reduced. Using this tagging mechanism the information of the arrival of the critical symbol C or D is propagated backwards in time.

5.3.2 Generalisation to SEM learning on multiple time-lags

In section 5.1 we proposed a time-lagged input presentation to capture temporal hidden causes. We noted that inference in this model suffered from a bias introduced by assuming successive network outputs to be conditionally independent. Interestingly, the importance sampling approach introduced for inferring HMM states, can also be used to overcome this problem. As in the HMM case, the bias arises from the normalisation of the WTA circuit. To correct the bias we can use importance weights, which we find from equations (5.6) and (5.5) to be given by

$$\begin{aligned} r(\mathbf{Z}) &= \frac{p(\mathbf{X}, \mathbf{Z})}{q(\mathbf{X} \mid \mathbf{Z})} \\ &= \prod_{\tau=1}^{T} \sum_{l=1}^{K} p\left(z_l^{(\tau)}\right) \prod_{m=1}^{M} p\left(\mathbf{x}^{(\tau-\Delta_m)} \mid z_l^{(\tau)}\right), \end{aligned} \tag{5.21}$$

which is again the product of all softmax denominators over the whole sequence, which can be tracked by a global inhibitory neuron. Thus, the same network architecture and weight tagging mechanism as introduced for the HMM case, can be used for the SEM with time-lags to correct the biased inference equations. Again we need to introduce the normalisation over L sampled paths to recover the conditional $p(\mathbf{Z} \mid \mathbf{X})$. We will cope with this problem in the next section.

5.3.3 Simplifications of the importance sampling approach

One obvious problem of the importance sampling approach is that a biologically plausible implementation of the weight update (5.18) and normalisation is not straightforward. In this section we will introduce a simplification of the model that addresses both problems.

Before we describe the exact formalism, let us first outline an environment that allows its implementation: In the discussion here we will restrict the online learning problems to the case, where the input is not given by an endless stream, but we have access to a process that chops the input into finite length sequences. This is not a very hard assumption since we have already seen in our discussion on statistical learning in section 2.3, that the detection of word boundaries seems to be a fundamental feature of the brain's sequence processing. Let us further assume that the sequences, found by the word boundary detection process, can be stored in short term memory and replayed, as discussed in section 2.2. The model we introduce here is thought to be located on top of this memory model, controlling its replay behaviour.

Assuming this input environment to be given, it is easy to redefine the sampling approach in a biologically plausible way, that avoids normalisation of the importance weights. To do so, we turn to the rejection sampling framework introduced in section 5.2. In a Monte Carlo framework we approximate the HMM sufficient statistics by drawing L hidden state sequences. Paths through the state space are generated by our network using the proposal distribution (5.11). To correct the bias introduce by not taking the backward messages into account it is sufficient, for the setup we have defined here, to use a binary weight deciding whether the sampled path is kept or not. If the path is kept the synaptic tags (5.16) get implemented into the weights, if the path is rejected the tags are simply discarded. This binary decision can be made upon the current value of the inhibitory unit $i^{(t)}$ given by equation (5.17). To see this we first note that $i^{(t)}$ is the probability of observing the current symbol, give the previous state variable

$$
\begin{aligned}
i^{(t)} &= \sum_{l=1}^{K} \mathrm{e}^{u_l^{(t)}} \\
&= \sum_{l=1}^{K} p\left(\mathbf{x}^{(t)} \mid z_k^{(t)}\right) p\left(z_k^{(t)} \mid \mathbf{z}^{(t-1)}\right) \\
&= p\left(\mathbf{x}^{(t)} \mid \mathbf{z}^{(t-1)}\right), \qquad (5.22)
\end{aligned}
$$

where we have used equations (5.17), (5.7) and the definition of the target weights (5.8). This quantity is a probability distribution over the current observation and can thus be used to make a decision, local in time, whether the current path should be accepted. The probability of accepting the path at any time step is given directly by the lateral inhibition. If in any time step, the circuit decides not to accept the path, the whole path is rejected and must be re-sampled using the short-term memory. The overall probability of accepting a path is given by the product over all local decisions

$$p\left(accept\right) = \prod_{\tau=1}^{T} p\left(\mathbf{x}^{(\tau)} \mid \mathbf{z}^{(\tau-1)}\right), \tag{5.23}$$

The overall joint probability of an accepted path $\mathbf{Z}$ and an input sequence $\mathbf{X}$ is given by the proposal distribution (5.11) and the probability of accepting it, given by

$$\begin{aligned} & q\left(\mathbf{Z} \mid \mathbf{X}\right) \times p\left(accept\right) \\ = & \frac{p\left(\mathbf{Z},\mathbf{X}\right)}{\prod_{\tau=1}^{T} \sum_{l=1}^{K} p\left(\mathbf{x}^{(\tau)} \mid z_l^{(\tau)}\right) p\left(z_l^{(\tau)} \mid \mathbf{z}^{(\tau-1)}\right)} \times \prod_{\tau=1}^{T} p\left(\mathbf{x}^{(\tau)} \mid \mathbf{z}^{(\tau-1)}\right) \\ = & \; p\left(\mathbf{Z},\mathbf{X}\right), \end{aligned} \tag{5.24}$$

which is the correct HMM joint probability. By sampling L paths using this framework we can approximate the HMM's sufficient statistics using similar calculations as for the importance sampling approach (5.15)

$$\xi\left(z_k^{(t)}, z_j^{(t-1)}\right) \approx \tfrac{1}{L} \textstyle\sum_{l=1}^{L} z_{k,l}^{(t)} \cdot z_{j,l}^{(t-1)}. \tag{5.25}$$

Here again, $z_{k,l}^{(t)}$ are the elements of the accepted paths. From this equation we can see that the normalisation in this framework is trivial since it is given by a constant $\frac{1}{L}$. Equation (5.25) converges to the correct HMM equation (3.12) for $L \to \infty$.

One obvious drawback of this approach is that the probability of accepting a path (5.23) can become small and thus re-sampling has to be performed very often. Especially if the paths become long this issue may cause problems. To improve the path acceptance rate we can introduce a normalising constant c into equation 5.23 as used in the original rejection sampling equation 3.8, such that

$$p\left(accept\right) = \prod_{\tau=1}^{T} c \cdot p\left(\mathbf{x}^{(\tau)} \mid \mathbf{z}^{(\tau-1)}\right). \tag{5.26}$$

In general selecting this constant will not be trivial since the maximum value of the inhibition is unknown and also depends on the current weights, which are changing during training. Obviously there is a tradeoff between using a small value for c, for which many paths will be rejected and thus sampling will take long, or using large values for which again a bias is introduced. If c is selected as large that every path is accepted, we restore the pure forward sampling results. In the next section we will show experimentally that in some practical cases good training results can be achieved by a very simple tracking mechanism for c and only using a single sampled path.

6 Numerical Experiments

In this chapter we will show the applicability of the extensions to the SEM network we have introduced in the previous chapter. We will investigate three different learning tasks: In the first task we will show that the time lagged extension introduced in section 5.1 is able to detect hidden causes from a spiking input stream. In the second task we will investigate the performance of the importance sampling approximation of HMMs from section 5.2 and compare it to standard HMM learning. We will show that it can achieve comparable results for a large class of problems. In the third experiment we will use the approximate HMM approach to learn artificial grammar models and compare the results with the behavioural study outlined in section 2.3. All experiments in this chapter were done using Matlab.

6.1 Learning spatiotemporal patterns using time-lagged inputs

In section 5.1 we have argued, that a SEM network, that receives input patterns on multiple time lags is able to detect temporal hidden causes. In this experiment we will confirm this by presenting sequences of spatiotemporal patterns to the network. Here, we use the biased inference equation (5.4) that treads the temporal hidden causes independently. We will see, that for the particular task used in this experiment, this simplified treatment is sufficient.

The patterns are fed into the network over 200 input neurons $x_i^{(t)}$ that create patterns with fixed spike times. The patterns were generated by drawing samples from a Poisson distribution with 15Hz rate. We created three patterns A, B and C, of length 50ms. Other than in previous experiments done in [Nessler et al., 2010] we did not present these patterns in random order, but only allowed fixed words A-B-C and A-C-B. These words were presented, with 50% probability each in an endless stream superposed with 5Hz random spike noise. Between two successive words, a 50ms frame with pure Poisson noise

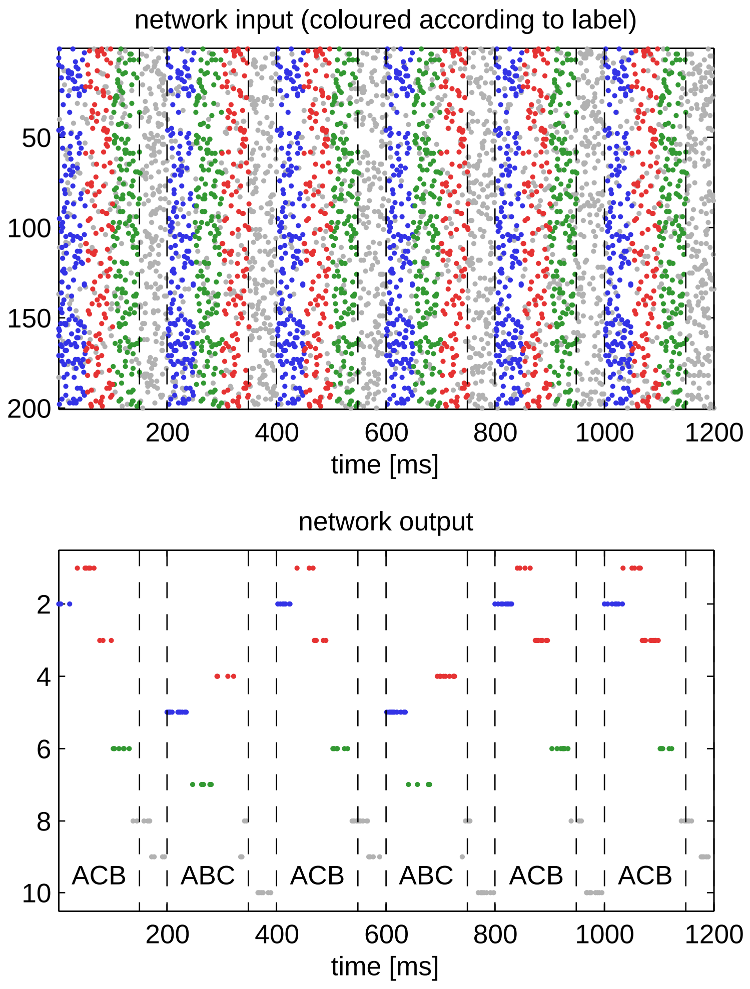


Figure 6.1: The input-output behaviour of the trained network. The top plot shows the network input spikes $\mathbf{x}^{(t)}$, for a single delay at 150ms. Spikes are coloured according to the pattern labels (blue: A, green: B, red: C, noise spikes in gray). The bottom plot shows the network output spikes, coloured according to the pattern identity during which the neurons are most prominently active. Word boundaries are indicated by dashed lines.

of 20Hz was presented. Thus, the overall input firing rate was 20Hz per input channel.

We used a network with $K = 10$ neurons to cluster this input, which was presented on four time lags: 0ms, 50ms, 100ms and 150ms. Throughout this experiment we used the extended learn rule for mixtures of Poisson distributions, outlined in section 4.4.3. The length of the rectangular EPSP window was chosen to $T = 30ms$. We let each SEM neuron fire with its individual Poisson-rate according to equation (4.12). The overall network activity was chosen to be $\lambda = 200Hz$. The weights were initialised to small, negative random values and trained using the learn rule for learning mixtures of Poisson distributions given in [Habenschuss, 2010].

We trained the network over 100 seconds. For the simulation we used fixed time steps of 1ms. Figure 6.1 shows the response of the trained network to an input sequence. The network responds with spike bursts at fixed locations within each pattern sequence. Without the time lagged inputs the cells would cluster the input into the three pattern identities A, B and C. With the simple extension we have presented here the network is able to recover the hidden cause variable of word identities. As expected, the network integrates information from different points in time. The neurons learn to distinguish a B inside a word A-B-C from one inside A-C-B. Each neuron, except those that adapted for the noise frames, have exclusively selected one combination of pattern and word identity. For example neuron seven responds with a spike bursts during pattern B, but only when the pattern appears inside the word A-B-C. If B appears inside A-C-B it remains silent. Neuron six shows the opposite behaviour. The cells, that emerged here using unsupervised learning, show similar properties as the context cells, outlined in section 2.1. Their selective behaviour is modulated by the current input history. The trained network is able to infer its output behaviour from temporal hidden causes.

6.2 Monte Carlo approximation of HMMs

In section 5.2 we introduced three approximations to the exact Baum-Welch HMM learning algorithm:

1. The pure forward sampling. The backward messages are completely ignored, the sampled paths are inferred from the forward messages only. This is the simplest learn rule for HMM learning proposed here, but the learn rule is biased and can lead in general to arbitrarily wrong results.

2. The second approach introduces the backward messages using importance sampling. It is the most efficient algorithm since all sampled paths are included in the estimate of the sufficient statistics, but its biological implementation is doubtful.

3. The rejection sampling approach is intermediate to the other approaches. It includes the backward messages through a stochastic process and a biological implementation is feasible.

In this section we will compare these three approaches in terms of their descriptive power using different learning tasks. For the comparison with standard HMM calculations we used the Hidden Markov Model (HMM) Toolbox for Matlab[1].

6.2.1 Convergence of the approximate algorithms

In the first experiment we verified the convergence of the Monte Carlo approaches for HMM learning to the exact forward-backward calculations. We applied the algorithms to approximate the E step of a HMM with 5 states and 10 observations. We generated random observation- and transition matrices by drawing them from a Beta distribution with $\alpha = 0.2$ and $\beta = 0.8$. We used this model to generate an observation sequence of length $T = 10$. Then we used equation (5.9) to draw state sequences from the model for the given observation. The target distribution was generated by computing the forward-backward algorithm on the sequence. We computed an approximate version of this distribution by using the importance- and rejection-sampling approaches. For the latter, the normalising constant was selected sufficiently small not to introduce a bias, for which we found the value $c = e^{1.25}$. To investigate the convergence properties of the algorithms we computed the Kullback-Leibler divergence between the sampled and the true distribution after each sampled path [Bishop, 2006]. Figure 6.2 shows the results. One can see that both algorithms converge asymptotically with one over the number of iterations to the target distribution.

[1] Hidden Markov Model (HMM) Toolbox for Matlab, written by Kevin Murphy, 1998. http://www.cs.ubc.ca/∼murphyk/Software/HMM/hmm.html

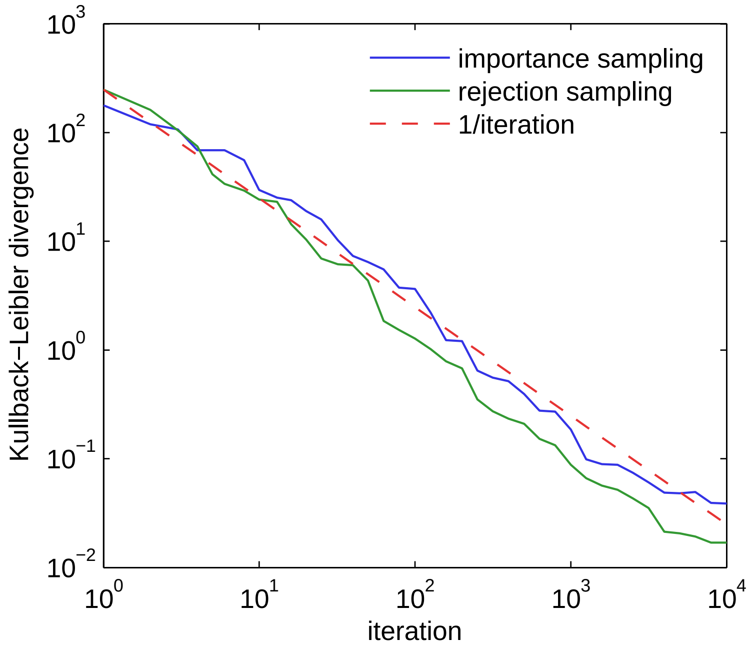


Figure 6.2: Demonstration of the Monte Carlo approximation for the forward-backward algorithm. Convergence of the Kullback-Leibler divergence between the sampled and the true distribution, with respect to the iteration is shown. Both, the importance-sampling and the rejection sampling approach converge asymptotically with one over the number of iterations.

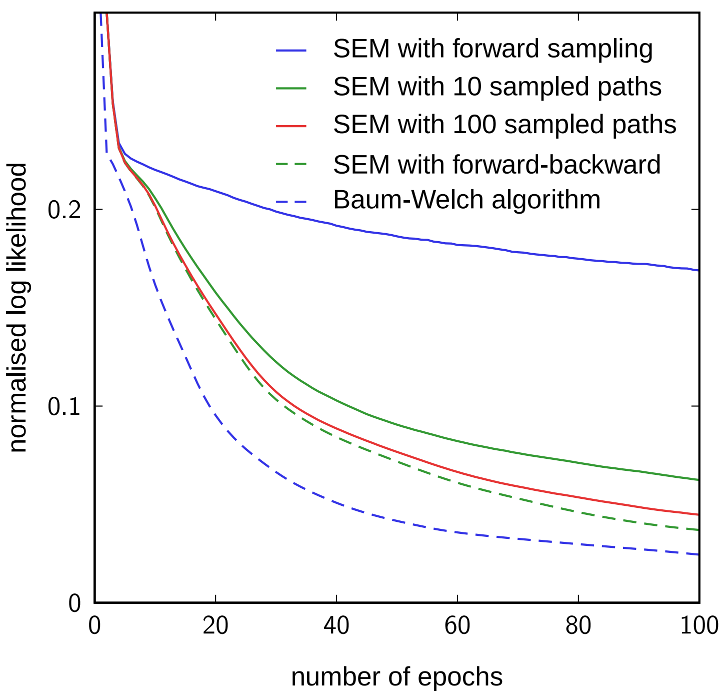


Figure 6.3: The mean of the normalised log likelihood errors of the different training approaches.

6.2.2 General HMM learning tasks

In order to give a quantitative comparison between the importance sampling introduced here and standard HMM learning, we applied it to solve a general HMM learning task. To do so we generated random teacher HMMs, with 5 hidden states and 10 observations and used them to generate observation sequences. Using this data we compared the training performance of five different methods: standard Baum Welch training on the whole batch of sequences, the online update rule (5.16) using the true sufficient statistics calculated using the forward-backward algorithm, forward sampling and the importance sampling approximation using 10 and 100 sampled paths.

The teacher HMMs were generated by drawing initial state, observation and transition probability tables from a Beta distribution with $\alpha = 0.2$ and $\beta = 0.8$ and then normalising the tables to proper conditional probabilities. The Beta prior on the parameters was chosen because it yields very rich and dynamic models, which allow a much better comparison of the learning performance than using a uniform prior. From each of these models we generated a train set of 200 and a test set of 2000 sequences of length $T = 50$.

For training we generated an initial model by drawing HMM parameters from a uniform distribution. We adapted this model to the train set using the five different training methods. The complete training data set was repeatedly present to the network. We will refer to the presentation of the whole batch of training sequence as an epoch. We applied the online learn rule after each sequence was observed and the Baum-Welch update after each epoch. The learn rate was chosen to be constant $\eta = 0.005$. We generated 50 trials using 50 different teacher HMMs. To get comparable results we used the normalised log-likelihood error [Mongillo and Deneve, 2008] given by

$$\lambda(i) = \frac{\langle \log \mathcal{L} \rangle (i) - \langle \log \mathcal{L} \rangle_{true}}{\langle \log \mathcal{L} \rangle_{init} - \langle \log \mathcal{L} \rangle_{true}} \tag{6.1}$$

where $\langle \log \mathcal{L} \rangle (i)$ is the average log likelihood over the test set given the current model at epoch i, $\langle \log \mathcal{L} \rangle_{init}$ is the average log likelihood of the initial model and $\langle \log \mathcal{L} \rangle_{true}$ the average log likelihood of the teacher model, that has created the data. This measure yields a normalised distance to the true model between zero and one.

The results of the five different training approaches are compared in figure 6.3. The figure shows the mean of the normalised log likelihoods over the 50 trials. As can be seen the Baum-Welch algorithm closely restored the teacher model and so did the online approach with the exact forward-backward calculations. Pure forward sampling, which is equal to ignoring the backward messages at all, was not sufficient to solve the task. But by sampling more and more paths we rapidly approach the performance of the exact calculations. The approximate training with 100 sampled paths already achieved a comparable result.

6.3 Artificial grammar learning

In the last experiment we investigate the ability of our network to learn *artificial grammar models*. Artificial grammars are, like grammars in natural language, models that define how symbols can be concatenated to form sequences. The models can be represented in a graph structure. An example grammar model, that was used in [Conway and Pisoni, 2008] is shown in figure 6.4. The model consists of 10 states, a set of six symbols {*1*, *2*, *3*, *4*, *Start*, *End*}, and transitions between the states. Each grammatical sequence begins with the *Start* symbol and terminates with the *End* symbol. In between any

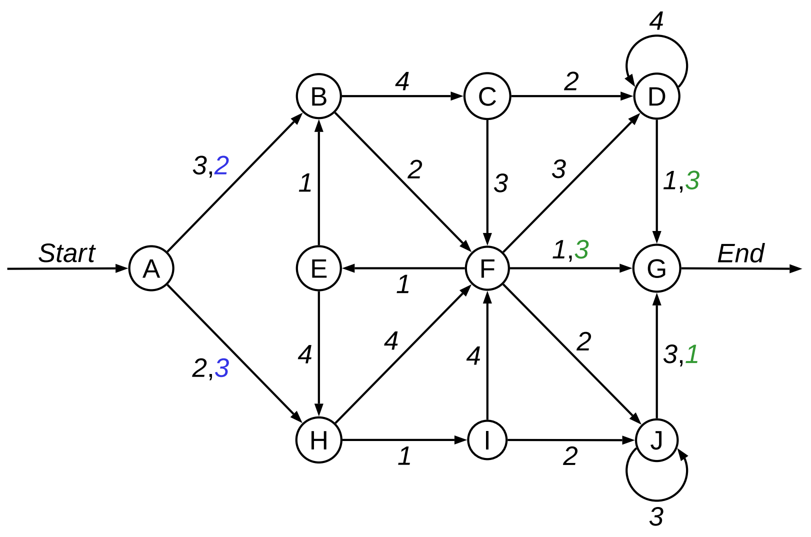


Figure 6.4: The artificial grammar model that was introduced in [Conway and Pisoni, 2008]. The model consists of 10 states indicated by circles and transitions between these states indicated by interlinking arcs. At each transition the symbol next to the arc is emitted. Besides the original grammar with symbols shown in black, two alternative grammars were created. One uses different symbols leaving node *A* shown in blue, the other uses different symbols entering node *G* shown in green.

path can be taken along the arcs that connect the states to generate a sequence. Outgoing arcs from the states represent possible transitions to the next state. In each time instant, one of the transitions is randomly selected and the next state is entered until the sequence terminates. At each transition being made the symbol that is associated with the transition is emitted. In that way the grammar model describes the set of possible and impossible sequences in a very compact way. For example a sequence *Start-2-4-1-End* is grammatical whereas *Start-2-4-4-End* is not under the given grammar, since there is no transition that emits symbol *4* outgoing from state F.

Obviously these grammar models are closely related to hidden Markov models. In fact it is very easy to find a HMM that represents the grammar shown here. The only difference between a HMM and a grammar model, is that symbols are emitted when a transition is made, and not when the state is entered[2].

[2]For the sake of completeness, it should be mentioned that some authors (e.g. [Stiller and Radons, 1999]) use an alternative definition of HMMs where symbols are generated on transitions. In our discussion on artificial grammar models, we will stick to the definition

train method	% correct classified	mean squared error
forward sampling	0.9310	0.0873
importance sampling	0.9550	0.0716

Table 6.1: The performance of the HMM approximation for the implicit learning task.

In this section we will investigate the capability of our model to learn such grammars.

6.3.1 Implicit learning of artificial grammars

In the first experiment on AGL, we used the small data set introduced in [Conway and Christiansen, 2005]. The data set consists of only 12 training sequences of lengths between three and six. As discussed in section 2.3 these sequences were played to participants in a memorisation phase. After that, the ability to distinguish between sequences generated from the same and a unknown grammar was investigated in a testing phase. In [Conway and Christiansen, 2005] 20 sequences were used for testing, ten of which were generated from the same grammar as for the train data (legal sequences $\mathcal{X}_l$) and ten from a different grammar (illegal sequences $\mathcal{X}_i$). In this experiment we repeated this task using the network introduced in the previous chapter. We created a network with six neurons and trained it using the forward sampling and the importance sampling learning. As in the original experiment each train sequence was presented twice. For training we used a constant learn rate $\eta = 0.05$ and five sampled paths for the importance sampling approach.

For the testing phase we used a threshold ϑ on the log-likelihood function to decide whether the sequence was legal. A sequence $\mathbf{X}$ was classified according to the function

$$c(\mathbf{X}, \rho) = \begin{cases} 1, & \text{if } \log \mathcal{L}(\mathbf{X}) \geq \vartheta \\ 0, & \text{else} \end{cases}, \tag{6.2}$$

where $c = 1$ indicates that the network decides that $\mathbf{X}$ is a legal sequence. Throughout these experiments we selected ϑ always to be the optimal threshold

we gave in the previous chapters, because it is more common and better fits our neural model. Also note that the two definitions are compatible and can be easily converted into each other.

that maximises the score function, which is given by

$$\vartheta^{*} = \arg\max_{\vartheta} \sum_{\mathbf{X}_l \in \mathcal{X}_l} c\left(\mathbf{X}_l, \vartheta\right) + \sum_{\mathbf{X}_i \in \mathcal{X}_i} \left(1 - c\left(\mathbf{X}_i, \vartheta\right)\right), \tag{6.3}$$

Note that in the original setup of the experiment we don't have access to the optimal threshold since its calculation requires access to the true class labels, but here we shall use it to compare the results in terms of the maximum achievable performance. To this end, the results we present here are an upper bound of the performance that can be achieved by the network. Further note that we have not yet defined how the network can evaluate the likelihood function. For the time being, we postpone these problems to future work.

We trained and tested the networks 50 times with random initial weights and averaged the classification score. The results are shown in table 6.1. Both train methods have reached near optimal score ($> 90\%$). Obviously, this very simple task can be solved without evaluation of the backward messages. The small networks of 6 neurons outperformed the results reported in the behavioural task of about $62\% - 75\%$ [Conway and Christiansen, 2005], but as discussed above, in a biologically realistic implementation of the classification, a considerable worse result is expected since access to the optimal threshold ϑ^{*} is not given.

6.3.2 Detailed models emerge from approximate HMM learning

We have outlined above that artificial grammar models are closely related to HMMs. A grammar model can be turned into a HMM by first inserting states at each arc, that emit the symbol associated with that arc. Then all states of the original grammar model must be removed and replaced with transitions between the HMM states. Some redundancy can be resolved, in a third step, by collapsing multiple states into one. This procedure results in a HMM with a sparse transition matrix, that is only non-zero for transitions associated with an arc. We found that our example grammar model from figure 6.4 can be turned into a HMM with 17 states, where four groups composed of four states encode the symbols *1-4* and one state encodes the *End* symbol. The *Start* symbol can be absorbed into the prior so we did not explicitly model a state that emits this symbol. We will refer to the resulting HMM as θ_{true}.

In total, the model is described by 38 non-zero prior and transition probabilities, which are listed as a set of transition rules in table 6.2(a). The rules define all possible transitions between the HMM states. Each line of the table defines all transitions that can be made leaving a certain state. The third line, for example, defines that states 1_c and 4_c can be reached from state 1_b. All transitions can be made with equal probability. The naming of the states is composed of the symbol they emit and a subscript letter to distinguish multiple states that emit the same symbol. By following a path through the state space it can be easily seen that the set of rules describes the same grammar as the graph in figure 6.4.

In this experiment we exploit the ability of the proposed neural network to restore the structure of θ_{true}. To do so we generated 10^6 grammatical sequences. Using this train set we trained a network with 17 neurons. The input layer was composed of five units encoding the symbols *1-4* and the *End* symbol. Again, we compare the performance of different training algorithms: pure forward sampling, importance sampling with five sampled paths and rejection sampling using a single sample. In order to get results that can be interpreted more easily we initialised the observation weights to the weights of the true observations of θ_{true} plus small equally distributed noise. Using this we assured that in the trained models each neuron encoded exactly one symbol. For training the networks we used the same parametrisation as in the experiment from section 6.2.2. For the rejection sampling algorithm we used a simple tracking of the normalising constant c, for which we used an exponential representation $c = \mathrm{e}^{\zeta}$. Whenever a path was accepted ζ was decreased by 10^{-4}, if the path was rejected ζ was increased by $5 \cdot 10^{-4}$. ζ was initialised to a relatively large value $\zeta = 2$, for which we found that, at the beginning of the learning every path was accepted. As learning proceeds ζ slowly converges to an equilibrium acceptance rate, which we found to be a feasible number of about eight samples per input sequence.

After the networks were trained, in the evaluation step, we tested the ability to classify unknown sequences to being grammatical or not. For this test we generated 10^3 legal sequences $\mathcal{X}_l$ from θ_{true}. In addition, we created a different grammar model from which we sampled a set of 10^3 illegal sequences $\mathcal{X}_i$. Obviously the performance will be correlated with the difference between the legal and illegal sequences, so we tested three different cases: A grammar θ_{rand} where each transition and emission was drawn randomly and two grammars that were created by modifying the original grammar model. The first one, θ_A was created by flipping the symbols leaving the first state A which is indicated in figure 6.4 in blue, and the second, θ_G was created by flipping the symbols entering the last state G shown in green. These three grammars differ in the location of the symbols that are different from the original model. θ_{rand} gen-

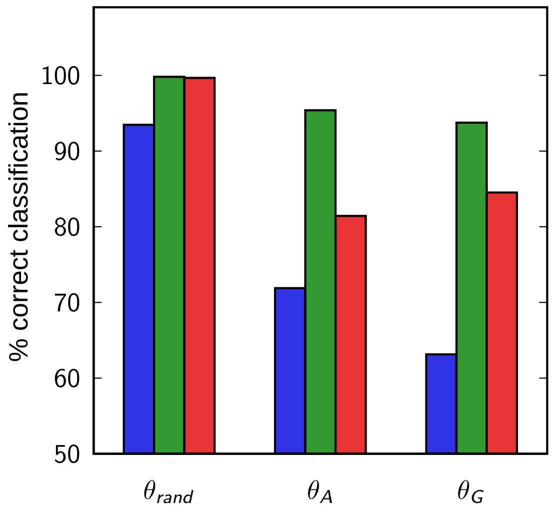


Figure 6.5: The percentage of correctly classified sequences for a network trained with forward sampling (blue), importance sampling (green) and rejection sampling (red), for the three different test grammars.

erates sequences that are different at all location whereas sequences generated by θ_A only differ at the beginning, that generated by θ_G only at the end. Using these test grammars, we again evaluated the classification score given the optimal threshold (6.3).

The results are depicted in figure 6.5. The figure shows the percentage of correctly classified sequences for the three different test grammars. The performance was averaged over 20 training results. It can be seen that five sampled paths were sufficient to achieve nearly perfect performance relatively independent of the used test grammar, for importance sampling. On the other hand the performance of the network with forward sampling showed strong dependence on the test grammar that was used. While it achieved comparable results for the random grammar, the performance dropped significantly for the two modified grammars. The rejection sampling approach achieved intermediate results, outperforming the forward sampling at all three cases.

As mentioned above, a rule based grammar model results in sparse transition probability tables. To get further insight why the learning approaches performed so differently we investigated up to what degree the rule-based structure was recovered. To do so we extracted the rules from the learned probability tables by accepting a transition rule for all entries that were significantly nonzero (≥ 0.01). The rules that were found in one exemplary training trial, by the network trained with importance sampling, are listed in table 6.2(b). As

(a) The grammatical rules for the artificial grammar model.

Start $\rightarrow 2_a, 3_a$
$1_a \rightarrow$ End
$1_b \rightarrow 1_c, 4_c$
$1_c \rightarrow 2_b, 4_a$
$1_d \rightarrow 2_d, 4_d$
$2_a \rightarrow 1_d, 4_d$
$2_b \rightarrow 1_a, 1_b, 2_d, 3_c$
$2_c \rightarrow 1_a, 4_b$
$2_d \rightarrow 3_d$
$3_a \rightarrow 2_b, 4_a$
$3_b \rightarrow 1_a, 1_b, 2_d, 3_c$
$3_c \rightarrow 1_a, 4_b$
$3_d \rightarrow 3_d$, End
$4_a \rightarrow 2_c, 3_b$
$4_b \rightarrow 1_a, 4_b$
$4_c \rightarrow 1_d, 4_d$
$4_d \rightarrow 1_a, 1_b, 2_d, 3_c$

(b) The grammar rules found by the neural network.

Start $\rightarrow 2_a, 3_a$
$1_a \rightarrow 2_b, 2_c, 4_a$
$1_b \rightarrow 1_a, 4_b$, End
$1_c \rightarrow$ End
1_d (*unreachable*)
$2_a \rightarrow 1_a, 4_a$
$2_b \rightarrow 1_b, 2_c, 3_b$
$2_c \rightarrow 1_b, 1_c, 2_c, 3_d, 4_c$
$2_d \rightarrow 1_b, 1_c, 4_c$
$3_a \rightarrow 2_b, 4_d$
$3_b \rightarrow 1_c, 3_d, 4_c$, End
$3_c \rightarrow 1_b, 1_c, 2_c, 3_c, 4_c$
$3_d \rightarrow 3_d$, End
$4_a \rightarrow 1_b, 2_c, 3_c$
$4_b \rightarrow 1_a, 4_a$
$4_c \rightarrow 1_c, 4_c$
$4_d \rightarrow 2_d, 3_c$

Table 6.2: The grammatical rules to describe the artificial grammar model from figure 6.4. The original model (a) and the model found using importance sampling (b). The states are named according to the symbols they emit, with a one-letter subscript if there are multiple states that emit the same symbol (of course the assignment of these letters is arbitrary).

mentioned above, the original model can be described with 38 rules. The SEM network selected 44 rules out of 306 possible transitions and thus recovered the sparse structure of the grammar model. By carefully comparing the two sets of rules we find many similarities. In fact eight out of 17 states have identical transition patterns. For most other states a counterpart in the original set can be found that shows at least some similarities. One big exception is the state 1_d which became reachable only with very low probability which resulted in an overall complexity increase of the model. But, although the set of rules is not perfect, a lot of the structure covered in the original grammar model was discovered.

On the other hand, the network trained with forward sampling did not capture the rule-based structure of the input at all. More than 200 transitions probabilities were found greater than 0.01, allowing transitions between almost every pair of states. The only significant structure that was captured is that no state that emits symbol *1* could be followed by a state that emits symbol *3*, and the *Start* state could only be followed by symbols *2* or *3*. By carefully inspecting the grammar model we see that a subsequence *1-3* is the only local transition that is not grammatical and obviously all sequences must start with *Start-2* or *Start-3*. Thus, the network trained without backward pass extracted only local statistics but largely ignored the global rule-based structure. Since the beginning of the sequence was modelled much more accurate than the end, by learning the correct local transitions leaving the *Start* state, we observed a significantly better performance for test grammar θ_A than θ_G. Again the rejection sampling approach achieved an intermediate result. The clear rule-based structure of the model was not recovered, but transition tables were found in general to be more structured. Apparently, the simple tracking of the normalising constant and sampling a single path is not sufficient to solve the learning task completely. But the results show that this simple implementation of rejection sampling significantly increased the quality of the learned models.

To test the impact of the network size on the classification performance we repeated the experiment with different numbers of output neurons. We generated six networks with different numbers of SEM units. Again, we initialised the observation matrices to the sparse structure encoding one preferred input symbol. Thus for each of the five inputs, a group of units showed increased response. We achieved this, like in the previous experiment by initialising the observation probabilities of each neuron k of one group l to $b_{ki} = \delta_{il} + \nu$, where ν is equally distributed noise between zero and 0.1. Each group encoding one input had the same size, except for the *End* symbol which was encoded by

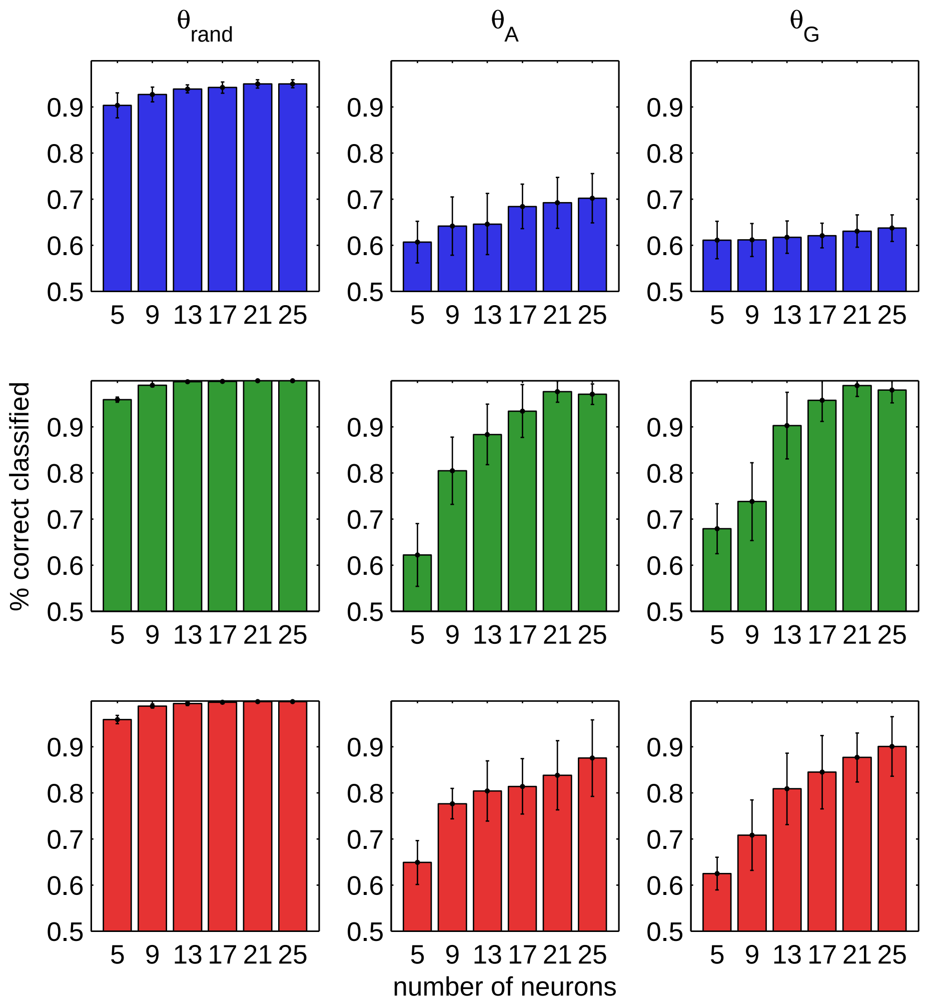


Figure 6.6: Comparison of the network performance for different network sizes using sparse initial weights. Forward sampling (blue), importance sampling (green) and rejection sampling (red).

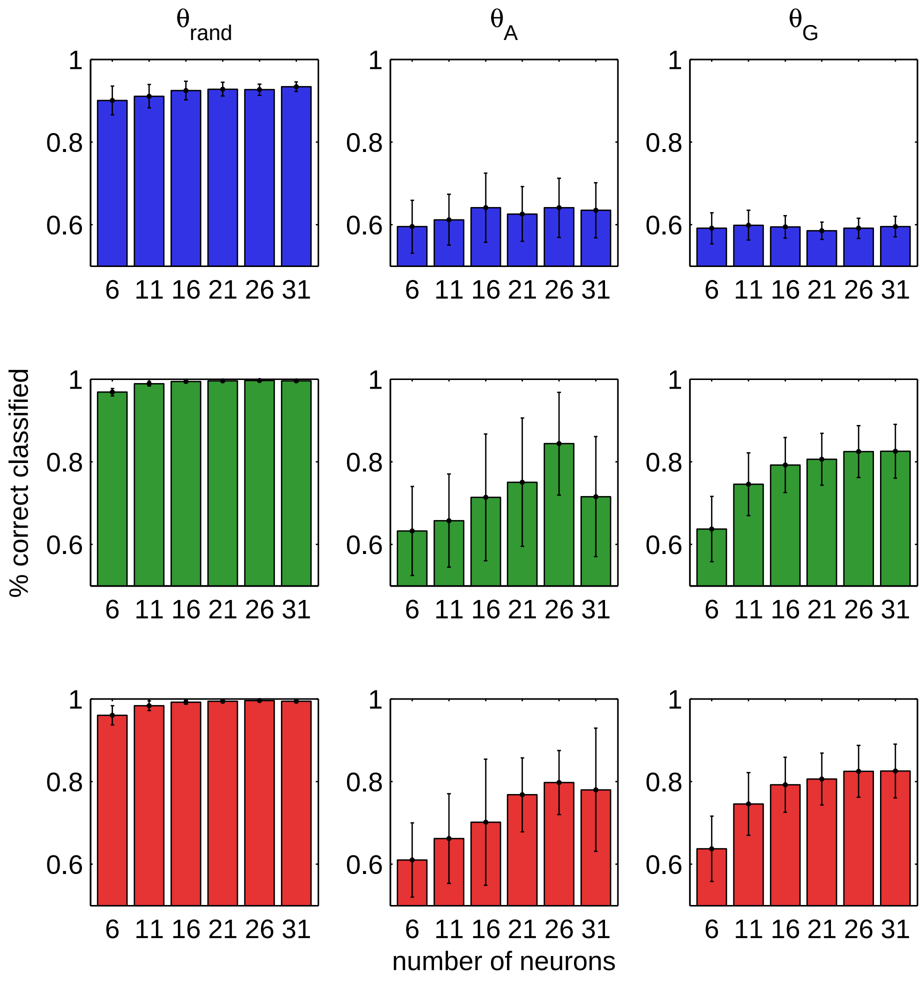


Figure 6.7: Comparison of the network performance for different network sizes using initial weights with Beta prior. Forward sampling (blue), importance sampling (green) and rejection sampling (red).

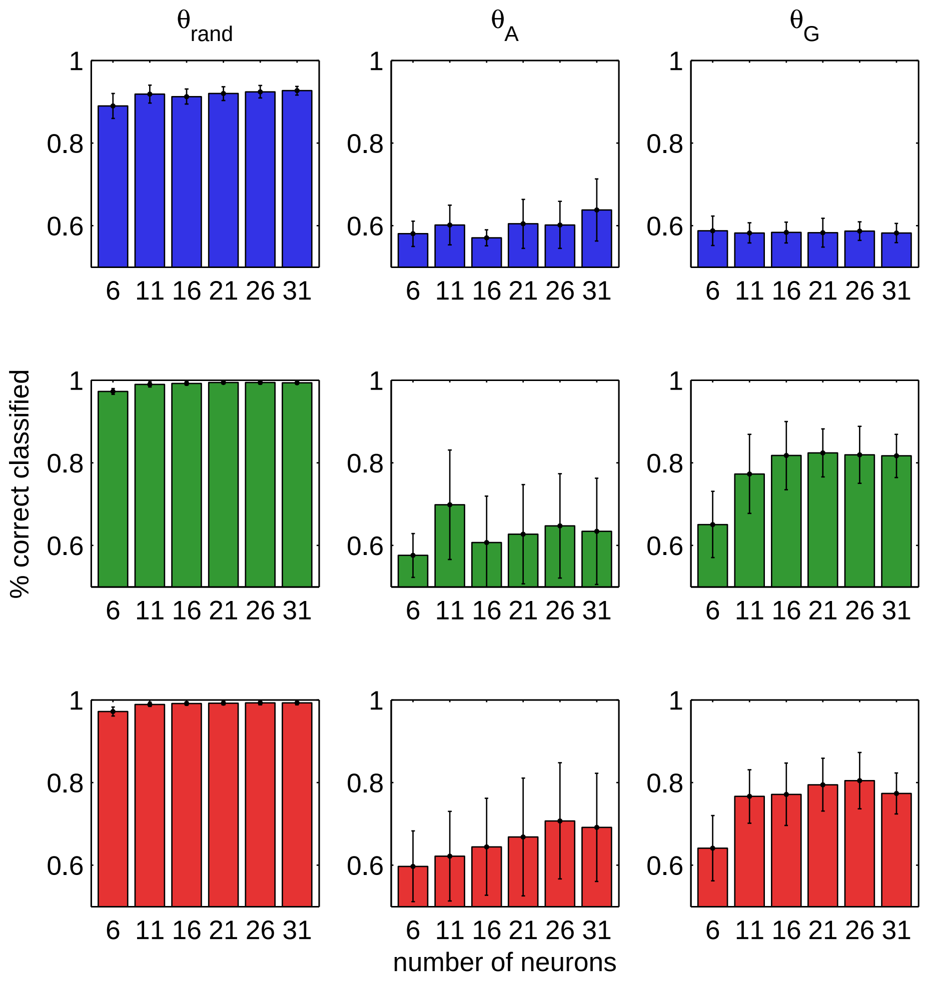


Figure 6.8: Comparison of the network performance for different network sizes using equally distributed initial weights. Forward sampling (blue), importance sampling (green) and rejection sampling (red).

only one cell. The networks were generated with increasing group size from 1 to 6, thus the first group had in total 5 units, the second 9, the third 13 and so on. We trained these networks using the same parametrisation as in the previous experiment. The results are shown in figure 6.3.2.

The performance of the networks trained with the three training approaches is shown. As can be seen the performance of the network trained without backward pass shows only little dependence on the network size. This can be explained by the fact that already a network with one neuron per symbol is sufficient to learn local statistics. However, the network trained with importance and rejection sampling showed a steep performance increase with the number of units. Not very surprisingly the network with six cells achieved about the same classification rate as the network without backward pass. The network with five neurons per symbols (21 in total), trained with importance sampling achieved almost perfect classification rates.

Obviously initialising the observation matrices to the sparse structure that was used throughout this experiment will also have an impact on the performance. In a third experiment we investigated the impact of the weight initialisation by repeating the above experiment, using initial transition matrices with a Beta prior ($\alpha = 0.2$, $\beta = 0.8$) and another initialisation with connectivity drawn from a uniform distribution. Again we trained networks of different size. Figures 6.3.2 and 6.3.2 shows the training results for Beta priors and random initialisation, respectively. Still we observe a significant increase in performance with the number of neurons, for the importance and rejection sampling approaches. But the results suggest, that the sparse structure has a large impact on the training result. The best performance is achieved using sparse and worst for equally distributed initial weights.

6.4 Discussion

We have seen, that when using pure forward sampling we ignore the complete backward information. The backward messages assure that the sampled paths, as a whole, are consistent with the model learned so far. Ignoring them, the model wildly jumps around between states that may explain the input data, loosing the global structure of the input, but still being able to capture local transition statistics. That explains why relatively good results were achieved for the random test grammar. In that case all the local statistical properties of the grammar were broken and even without the exact knowledge of the grammar rules, the SEM network was able to distinguish random sequences from grammatical ones. This accounts for the concept of statistical learning, outlined in section 2.3. In the early learning phase only a rudimentary representation of the signal statistics is formed, but the global rule-based structure is ignored. We have seen that forward sampling alone is sufficient to explain data from behavioural tasks [Conway and Christiansen, 2005], but it is not able to learn detailed models.

The importance sampling approaches, on the other hand, gain access to the backward messages at least approximately, which enables them to extract more information than just local statistics. This allows the model to discover much more detail of the grammar model. The evaluation of the importance weights can be interpreted as a form of hypothesis verification. Paths that are sampled but contradict the model learned so far, obtain a weight close to zero and are thus discarded and not used for the learning. Due to this hypothesis verification strategy the network gains access to an almost perfect rule-like structure of the grammar model, which results in the sparse structure of the transition probability table.

Thus we have seen, that statistical learning and the discovery of rule-based structures, can be grounded on one common neural model. These findings provide evidence that the two mechanisms are not mutually exclusive but might be successive strategies of model refinement. First statistical learning extracts some statistical properties of the input and then, as the model gets more and more accurate, the hypothesis verification sets in to extract the fine structure.

7 Conclusion and future work

In this thesis we investigated two extensions of the SEM network to capture temporal hidden causes, that emerged from extending the input. The first one uses dendritic delays to integrate input statistics at multiple time lags. The second uses lateral connections to include the previous time step when inferring the current state. We have seen that both approaches give rise to one common problem: Exact inference of the hidden causes depends on the whole input sequence, since the independence assumption is broken when temporal hidden causes are involved. Nevertheless, we have seen that still assuming the hidden causes to be independent in a forward sampling manner, is sufficient to explain some experimental results: the emergence of context cells [Barone and Joseph, 1989] and statistical learning [Perruchet and Pacton, 2006].

We introduced two Monte Carlo approximations of the exact inference that correct the bias of the forward sampling. Both algorithms need to extend the basic theory of STDP learning using a synaptic tagging mechanism. In fact, the results presented here suggest, that classical STDP learning is not adequate to learn the parametrisation of models involving temporal hidden causes, since exact inference always needs to take future events into account. We solved this problem of looking into the future here, by delaying the consolidation of the weight changes until the whole sequence is read to its end. During this temporal window an inhibitory neuron, that acts as a global observer, is able to interfere with the weight consolidation by modulating the present synaptic tags. Biological data suggests that there are at least two neural mechanisms how this modulation could be realised: One mechanism involves controlled release of dopamine, the other uses specific activity patterns to actively reset or consolidate the synaptic tags [Frey et al., 1990, Otani et al., 2003, Young and Nguyen, 2005].

The two approaches we have proposed are appealing from different perspectives: The importance sampling approach is preferable from a computational point of view, since no sampled path needs to be rejected completely and thus sampling is more efficient. But, we have argued that a neural implementation is doubtful. The rejection sampling framework for HMMs we have presented here, results in very compact mathematics based only on binary decisions, local both in space and time. Also the normalisation over all importance weights is not an issue in this context, since the probabilities of accepting one path are simply normalised by replaying the same input until one single path is accepted. This means that the number of times a sequence is replayed is proportional to $\frac{1}{p(\mathbf{X})}$. Thus, sequences that are novel and to that fact not well represented in the learned model get more attention in the learning process, which is an intuitive assumption and supported by experimental data [Peyrache et al., 2009]. It must be stressed however that, although there are reports on neural replay of complex patterns from memory, the functional role we propose here and the connection to synaptic tagging, arises only from theoretical considerations.

So far, paths were sampled independent of each other, which could result in sampling the same path multiple times even if it had been rejected. In a more advanced setup one could use the local information of synaptic tags and early LTP to modulate the probability distribution for subsequent samples, leading to a Markov chain Monte Carlo approach. Such an approach would lead to more efficient sampling and might work without a normalising constant we found hard to select.

In the experiments on AGL we computed the likelihood function using standard HMM calculations. In a biologically realistic setup however a direct evaluation by the network itself would be desirable. Again, the inhibitory neuron might play a major role in this evaluation, since we have seen that the probability of accepting a path is proportional to the path likelihood. Future work will have to show, whether a reliable decision can be made solely based on calculations performed by the inhibitory unit.

The two extensions to the basic SEM we have proposed are not orthogonal to each other. Since they only emerged from extending the network input, a network that uses both, dendritic delays and lateral connections would be feasible. Also note that the exclusive activity of an external unit at the beginning of a sequence, which we used to model the initial probabilities of HMMs, would easily enable the model to form hierarchies. Single cells of a WTA circuit could act as initial units for other circuits. Taking all together large neural models could be build with multiple hierarchically organised SEM circuits integrating information over long time windows.

Bibliography

[Alamino and Caticha, 2008] Alamino, R. and Caticha, N. (2008). Bayesian online algorithms for learning in discrete hidden Markov models. *Discrete and Continuous Dynamical Systems Series B*, 9(1):1–10. (Cited on page 25.)

[Aronov et al., 2008] Aronov, D., Andalman, A., and Fee, M. (2008). A specialized forebrain circuit for vocal babbling in the juvenile songbird. *Science*, 320:630–633. (Cited on pages 10 and 11.)

[Averbeck et al., 2002] Averbeck, B., Chafee, M., Crowe, D., and Georgopoulos, A. (2002). Parallel processing of serial movements in prefrontal cortex. *Proceedings of the National Academy of Sciences*, 99(20):13172–13177. (Cited on pages 2, 6 and 7.)

[Baldi and Chauvin, 1994] Baldi, P. and Chauvin, Y. (1994). Smooth online learning algorithms for hidden Markov models. *Neural Computation*, 6(2):307–318. (Cited on pages 24 and 45.)

[Barone and Joseph, 1989] Barone, P. and Joseph, J. (1989). Prefrontal cortex and spatial sequencing in macaque monkey. *Experimental Brain Research*, 78(3):447–464. (Cited on pages 2, 5, 6 and 79.)

[Baum and Petrie, 1966] Baum, L. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563. (Cited on pages 1, 20 and 23.)

[Bengio, 1999] Bengio, Y. (1999). Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162. (Cited on page 18.)

[Berdyyeva and Olson, 2009] Berdyyeva, T. and Olson, C. (2009). Monkey supplementary eye field neurons signal the ordinal position of both actions and objects. *The Journal of Neuroscience*, 29(3):591–599. (Cited on page 7.)

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer (New York). (Cited on pages 1, 13, 16, 17, 18, 23, 29, 43 and 62.)

[Bliss and Lømo, 1973] Bliss, T. and Lømo, T. (1973). Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. *The Journal of Physiology*, 232(2):331–356. (Cited on page 31.)

[Bobrowski et al., 2009] Bobrowski, O., Meir, R., and Eldar, Y. (2009). Bayesian filtering in spiking neural networks: Noise, adaptation, and multisensory integration. *Neural computation*, 21(5):1277–1320. (Cited on page 38.)

[Bolhuis and Gahr, 2006] Bolhuis, J. and Gahr, M. (2006). Neural mechanisms of birdsong memory. *Nature Reviews Neuroscience*, 7(5):347–357. (Cited on pages 10 and 11.)

[Buzsáki, 1989] Buzsáki, G. (1989). Two-stage model of memory trace formation: A role for 'noisy' brain states. *Neuroscience*, 31(3):551–570. (Cited on page 8.)

[Caporale and Dan, 2008] Caporale, N. and Dan, Y. (2008). Spike timing-dependent plasticity: A hebbian learning rule. *The Annual Review of Neuroscience*, 31:25–46. (Cited on page 31.)

[Churchill, 1989] Churchill, G. (1989). Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51(1):79–94. (Cited on page 2.)

[Cleeremans et al., 1998] Cleeremans, A., Destrebecqz, A., and Boyer, M. (1998). Implicit learning: News from the front. *Trends in cognitive sciences*, 2(10):406–416. (Cited on pages 2 and 9.)

[Conway and Christiansen, 2005] Conway, C. and Christiansen, M. (2005). Modality-constrained statistical learning of tactile, visual, and auditory sequences. *Journal of Experimental Psychology*, 31(1):24–39. (Cited on pages 9, 67, 68 and 77.)

[Conway and Pisoni, 2008] Conway, C. and Pisoni, D. (2008). Neurocognitive basis of implicit learning of sequential structure and its relation to language processing. *Annals of the New York Academy of Sciences*, 1145:113–131. (Cited on pages 9, 65 and 66.)

[Dan and Poo, 2004] Dan, Y. and Poo, M. (2004). Spike timing-dependent plasticity of neural circuits. *Cell Press*, 44:23–30. (Cited on page 31.)

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B*, 39:1–22. (Cited on page 15.)

[Deneve, 2008a] Deneve, S. (2008a). Bayesian spiking neurons I: Inference. *Neural Computation*, 20:91–117. (Cited on page 38.)

[Deneve, 2008b] Deneve, S. (2008b). Bayesian spiking neurons II: Learning. *Neural Computation*, 20:118–145. (Cited on page 39.)

[Douglas and Martin, 2004] Douglas, R. and Martin, K. (2004). Neuronal circuits of the neocortex. *The Annual Review of Neuroscience*, 27(1):419–451. (Cited on page 34.)

[Euston et al., 2007] Euston, D., Tatsuno, M., and McNaughton, B. (2007). Fast-forward playback of recent memory sequences in prefrontal cortex during sleep. *Science*, 318:1147–1150. (Cited on page 9.)

[Frey and Morris, 1998] Frey, U. and Morris, R. (1998). Synaptic tagging: implications for late maintenance of hippocampal long-term potentiation. *Trends in neurosciences*, 21(5):181–188. (Cited on page 32.)

[Frey et al., 1990] Frey, U., Schroeder, H., Matthies, H., et al. (1990). Dopaminergic antagonists prevent long-term maintenance of posttetanic LTP in the CA1 region of rat hippocampal slices. *Brain research*, 522(1):69–75. (Cited on pages 32 and 79.)

[Froemke and Dan, 2002] Froemke, R. and Dan, Y. (2002). Spike-timing-dependent synaptic modification induced by natural spike trains. *Nature*, 416:433–438. (Cited on page 31.)

[Gerstner and Kistler, 2002] Gerstner, W. and Kistler, W. (2002). *Spiking neuron models*, volume 15. Cambridge University Press Cambridge, UK. (Cited on pages 27, 29, 30, 31 and 34.)

[Gray and Singer, 1989] Gray, C. and Singer, W. (1989). Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex. *Proceedings of the National Academy of Sciences of the United States of America*, 86(5):1698–1702. (Cited on page 34.)

[Habenschuss, 2010] Habenschuss, S. (2010). Novel methods for probabilistic inference and learning in spiking neural networks. Master's thesis, Graz University of Technology, A-8010 Graz, Austria. (Cited on pages 37 and 61.)

[Hahnloser and Kotowicz, 2010] Hahnloser, R. and Kotowicz, A. (2010). Auditory representations and memory in birdsong learning. *Current Opinion in Neurobiology*, 20:332–339. (Cited on pages 10 and 11.)

[Hebb, 1949] Hebb, D. (1949). The organisation of behaviour. Wiley, New York. (Cited on page 30.)

[Hernández et al., 1996] Hernández, L., Moral, S., and Salmerón, A. (1996). Importance sampling algorithms for belief networks based on approximate computation. In *Proceedings of the IPMU'96 Conference*, volume II, pages 859–864. (Cited on page 16.)

[Huda et al., 2009] Huda, S., Yearwood, J., and Togneri, R. (2009). A stochastic version of expectation maximization algorithm for better estimation of hidden markov model. *Pattern Recognition Letters*, 30:1301–1309. (Cited on page 25.)

[Isomura et al., 2006] Isomura, Y., Sirota, A., Özen, S., Montgomery, S., Mizuseki, K., Henze, D., and Buzsáki, G. (2006). Integration and segregation of activity in entorhinal-hippocampal subregions by neocortical slow oscillations. *Neuron*, 52(5):871–882. (Cited on pages 8 and 9.)

[Izhikevich, 2007] Izhikevich, E. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, 17(10):2443–2452. (Cited on pages 32 and 50.)

[Juang and Rabiner, 1990] Juang, B. and Rabiner, L. (1990). The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(9):1639–1641. (Cited on page 24.)

[Konishi, 1965] Konishi, M. (1965). The Role of Auditory Feedback in the Control of Vocalization in the White-Crowned Sparrow. *Zeitschrift für Tierpsychologie*, 22(7):770–783. (Cited on page 10.)

[Krishnamurthy and Moore, 1993] Krishnamurthy, V. and Moore, J. B. (1993). On-line estimation of hidden Markov model parameters based on the Kullback-Leibler information measure. *IEEE Transactions on Signal Processing*, 41:2557–2573. (Cited on page 25.)

[Krug et al., 1984] Krug, M., Lössner, B., and Ott, T. (1984). Anisomycin blocks the late phase of long-term potentiation in the dentate gyrus of freely moving rats. *Brain research bulletin*, 13(1):39–42. (Cited on page 32.)

[Lisman, 1989] Lisman, J. (1989). A mechanism for the Hebb and the anti-Hebb processes underlying learning and memory. *Proceedings of the National Academy of Sciences of the United States of America*, 86(23):9574–9578. (Cited on page 31.)

[Maass, 2000] Maass, W. (2000). On the computational power of winner-take-all. *Neural Computation*, 12(11):2519–2535. (Cited on page 34.)

[Maass and Bishop, 2001] Maass, W. and Bishop, C. (2001). *Pulsed neural networks*. The MIT Press. (Cited on page 33.)

[MacKay, 2003] MacKay, D. (2003). *Information theory, inference, and learning algorithms*. Cambridge Univ Pr. (Cited on pages 15 and 17.)

[Markram et al., 1997] Markram, H., Lubke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275:213–215. (Cited on page 31.)

[McCulloch and Pitts, 1943] McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 5(4):115–133. (Cited on page 28.)

[Merhav and Ephraim, 1991] Merhav, N. and Ephraim, Y. (1991). Maximum likelihood hidden Markov modeling using a dominant sequenceof states. *IEEE Transactions on Signal Processing*, 39(9):2111–2115. (Cited on page 25.)

[Mongillo and Deneve, 2008] Mongillo, G. and Deneve, S. (2008). Online learning with hidden Markov models. *Neural Computation*, 20:1706–1716. (Cited on pages 24, 39 and 65.)

[Nessler et al., 2010] Nessler, B., Pfeiffer, M., and Maass, W. (2010). STDP enables spiking neurons to detect hidden causes of their inputs. In *Proceedings of NIPS 2009: Advances in Neural Information Processing Systems*, volume 22, pages 1357–1365. MIT Press. (Cited on pages 2, 35, 36, 37, 41, 49 and 59.)

[Nielsen, 2000] Nielsen, S. F. (2000). The stochastic EM algorithm: estimation and asymptotic results. *Bernoulli*, 6(3):457–489. (Cited on page 17.)

[O'Keefe and Recce, 1993] O'Keefe, J. and Recce, M. (1993). Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3):317–330. (Cited on page 34.)

[Otani et al., 2003] Otani, S., Daniel, H., Roisin, M., and Crepel, F. (2003). Dopaminergic modulation of long-term synaptic plasticity in rat prefrontal neurons. *Cerebral Cortex*, 13(11):1251–1256. (Cited on pages 32 and 79.)

[Perruchet and Pacton, 2006] Perruchet, P. and Pacton, S. (2006). Implicit learning and statistical learning: One phenomenon, two approaches. *Trends in Cognitive Sciences*, 10(5):233–238. (Cited on pages 9 and 79.)

[Peyrache et al., 2009] Peyrache, A., Khamassi, M., Benchenane, K., Wiener, S., and Battaglia, F. (2009). Replay of rule-learning related neural patterns in the prefrontal cortex during sleep. *Nature Neuroscience*, 12(7):919–926. (Cited on pages 8, 9 and 80.)

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286. (Cited on pages 2, 18 and 23.)

[Reber, 1967] Reber, A. (1967). Implicit learning of artificial grammars. *Journal of verbal learning and verbal behavior*, 6:855–863. (Cited on page 9.)

[Roediger et al., 2007] Roediger, H., Dudai, Y., and Fitzpatrick, S. (2007). *Science of memory: concepts*. Oxford University Press, USA. (Cited on page 8.)

[Saffran et al., 1996] Saffran, J., Aslin, R., and Newport, E. (1996). Statistical learning by 8-month-old infants. *Science*, 274:1926–1928. (Cited on page 9.)

[Sajikumar and Frey, 2004] Sajikumar, S. and Frey, J. (2004). Resetting of ‘synaptic tags’ is time- and activity-dependent in rat hippocampal CA1 in vitro. *Neuroscience*, 129(2):503–507. (Cited on page 33.)

[Salinas, 2009] Salinas, E. (2009). Rank-order-selective neurons form a temporal basis set for the generation of motor sequences. *The Journal of Neuroscience*, 29(14):4369–4380. (Cited on page 7.)

[Sirota et al., 2003] Sirota, A., Csicsvari, J., Buhl, D., and Buzsáki, G. (2003). Communication between neocortex and hippocampus during sleep in rodents. *Proceedings of the National Academy of Sciences of the United States of America*, 100(4):2065–2069. (Cited on page 8.)

[Softky and Koch, 1993] Softky, W. and Koch, C. (1993). The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs. *The Journal of Neuroscience*, 13(1):334–350. (Cited on page 33.)

[Stiller and Radons, 1999] Stiller, J. C. and Radons, G. (1999). Online estimation of hidden Markov models. *IEEE Signal Processing Letters*, 6(8):213–215. (Cited on pages 24 and 66.)

[Thrun et al., 1999] Thrun, S., Langford, J., and Fox, D. (1999). Monte Carlo hidden Markov models: Learning non-parametric models of partially observable stochastic processes. In *Proceedings of the 16th International Conference on Machine Learning (ICML'99)*, pages 415–424. (Cited on page 25.)

[Vates and Nottebohm, 1995] Vates, G. and Nottebohm, F. (1995). Feedback circuitry within a song-learning pathway. *Proceedings of the National Academy of Sciences of the United States of America*, 92(11):5139–5143. (Cited on page 10.)

[Viterbi, 1967] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269. (Cited on page 25.)

[von der Malsburg and Buhmann, 1992] von der Malsburg, C. and Buhmann, J. (1992). Sensory segmentation with coupled neural oscillators. *Biological Cybernetics*, 67(3):233–242. (Cited on page 34.)

[Yamagishi, 2006] Yamagishi, J. (2006). An introduction to HMM-based speech synthesis. Technical report, Tokyo Institute of Technology. (Cited on page 18.)

[Young and Nguyen, 2005] Young, J. and Nguyen, P. (2005). Homosynaptic and heterosynaptic inhibition of synaptic tagging and capture of long-term potentiation by previous synaptic activity. *The Journal of Neuroscience*, 25(31):7221–7231. (Cited on pages 33 and 79.)

[Zemel et al., 1998] Zemel, R., Dayan, P., and Pouget, A. (1998). Probabilistic interpretation of population codes. *Neural Computation*, 10(2):403–430. (Cited on page 34.)