GRAZ UNIVERSITY OF TECHNOLOGY

DISSERTATION

to obtain the title of

Doctor of Technical Sciences

of Graz University of Technology

Graz University of Technology

Defended by

Dejan Aleksandar Pecevski

Modelling Inference and Learning in Biological Networks of Neurons

Thesis Advisor: O.Univ. Prof. DI Dr.rer.nat. Wolfgang Maass

defended on October, 7th, 2011

Jury:

Advisor:	O.Univ. Prof. DI Dr.rer.nat. Wolfgang Maass	-	TU Graz
Reviewer:	Univ. Prof. Dr.rer.nat. Gordon Pipa	-	Osnabrück Univ.
Dean of Studies:	Assoc. Prof. DI Dr.tech. Oswin Aichholzer	-	TU Graz

Eidesstattliche Erklärung

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

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.

Graz, May 2011

....
(signature)

Abstract

In this thesis top-down and bottom-up methods are applied in the study of two central questions regarding the function of neural circuits in the brain: what type of computations they implement and how learning on a synaptic level yields useful computational functions on a circuit and behavioral level. Motivated by the need to best support the neural modelling and simulation requirements of the research done in this thesis and other related work, a novel software framework for neural simulations named PCSIM was developed, which is another additional contribution of this thesis.

Probabilistic inference in graphical models has been often proposed as a suitable candidate framework for explaining the computations that the brain carries out, but the neural basis of these computations remains unclear. In chapter 2 this problem is approached, and several different possible implementations of probabilistic inference in graphical models with networks of spiking neurons are presented. The developed neural implementations perform probabilistic inference through Markov chain Monte Carlo sampling and use specific network structures or dendritic computations in biologically realistic neurons as basic building blocks to realize the required nonlinear computational operations. Hence, they propose that the computational function of local network motifs as well as the dendritic computations in single neurons is to support the probabilistic inference operations on a larger network level.

In chapter 3 it is analysed theoretically and through computer simulations what computations can be learned with reward-modulated spike-timing-dependent plasticity, a synaptic plasticity learning rule based on experimental findings about long-term synaptic efficacy changes. In particular, it is shown that this plasticity rule enables spiking neurons to learn classification of temporal spike patterns. It is also shown that neurons can learn with this rule a specific mapping from input spike patterns to output spike patterns. Moreover, it is analysed under which conditions and parameters values for the learning rule and the neuron model the learning in these learning tasks is successful. Finally, it is also demonstrated that reward-modulated STDP can explain experimental results on biofeedback learning in monkeys.

Chapter 4 gives an overview of the *P*arallel neural *C*ircuit *SIM*ulator (PCSIM) with a focus on its integration with the Python programming language. PCSIM is a neural simulation environment intended for simulation of spiking and analog neural networks with a support for distributed simulation of large-scale neural networks on multiple machines. In this chapter key features of PCSIM's modular and extensible object-oriented framework and user interface are outlined and it is described how these features enable the user to develop and construct neural models easier and faster, to speed up the simulations of the models, and to add easily custom extensions to the PCSIM framework. Further, benefits from the integration of PCSIM with Python are elucidated.

Keywords: probabilistic inference, sampling, graphical models, spiking neurons, network motifs, dendritic processing, reward-modulated spike-

timing-dependent plasticity, STDP, reward-based learning, biofeedback, PCSIM, neural simulator, parallel simulation, Python

Zusammenfassung

In dieser Dissertation werden top-down und bottom-up Methoden zur Untersuchung von zwei zentralen Fragen herangezogen, die die Funktion von neuralen Schaltkreisen im Gehirn betreffen: welche Art von Berechnungen sie implementieren und wie Lernen auf synaptischer Ebene zu sinnvollen Berechnungsfunktionen auf Schaltkreis- und Verhaltensebene führt. Motiviert durch den Bedarf einer bestmöglichen Unterstützung der Anforderungen hinsichtlich neuronaler Modellierung und Simulation an die Forschung, die in dieser Dissertation und verwandten Arbeiten durchgeführt wurde, wurde ein neues Software-Framework für neuronale Simulationen namens PCSIM entwickelt, das einen weiteren zusätzlichen Betrag dieser Dissertation darstellt.

Probabilistische Inferenz in graphischen Modellen ist oft als ein geeigneter Kandidat für ein Framework zur Erklärung der Berechnungen vorgeschlagen worden, die das Gehirn ausführt, die neurale Basis dieser Berechnungen blieb jedoch unklar. In Kapitel 2 wird an dieses Problem herangegangen, und eine Reihe von verschiedenen möglichen Implementationen von probabilistischer Inferenz in graphischen Modellen in Netzwerken von spikenden Neuronen werden präsentiert. Die entwickelten neuronalen Implementationen führen probabilistische Inferenz mittels Markov Chain Monte Carlo-Sampling aus und verwenden spezifische Netzwerkstrukturen oder dendritische Berechnungen in biologisch realistischen Neuronen als elementare Bausteine zur Realisierung der notwendigen nichtlinearen Berechnungsoperationen. Das deutet darauf hin, dass die Berechnungsfunktionalität von lokalen Netzwerkmotiven sowie die dendritischen Berechnungen in einzelnen Neuronen Operationen für probabilistische Inferenz auf einer höheren Netzwerkebene unterstützen sollen.

In Kapitel 3 wird theoretisch und durch Computersimulationen analysiert, welche Berechnungen mit belohnungsmodulierter Spike-Timing-Dependent Plasticity gelernt werden können, einer Lernregel für synaptische Plastizität, die auf experimentellen Erkenntnissen über langfristige änderungen der synaptischen Wirksamkeit beruht. Insbesondere wird gezeigt, dass diese Plastizitätsregel es spikenden Neuronen erlaubt, Klassifikationen von temporalen Spike-Mustern zu lernen. Es wird ebenfalls gezeigt, dass Neuronen mit dieser Regel eine spezifische Abbildung von Input- auf Output-Spike-Mustern lernen können. Darüberhinaus wird analysiert, unter welchen Bedingungen und Parametern für die Lernregel und das Neuronenmodell das Lernen in diesen Aufgabenstellungen erfolgreich ist. Abschließend wird demonstriert, dass belohnungsmodulierte STDP experimentelle Resultate des Lernens von Biofeedback in Affen erklären kann.

Kapitel 4 bietet einen überblick über den parallelen neuralen Schaltkreissimulator PCSIM (*P*arallel neural *C*ircuit *SIM*ulator) mit dem Schwerpunkt auf dessen Integration mit der Python Programmiersprache. PCSIM ist eine neuronale Simulationsumgebung, die für die Simulation von spikenden und analogen neuronalen Netzwerken vorgesehen ist, und die verteilte Simulationen von großen neuronalen Netzwerken auf mehreren Maschinen unterstützt. In diesem Kapitel werden die Hauptmerkmale des modularen und erweiterbaren objektorientierten Frameworks und User-Interfaces vorgestellt, und es wird beschrieben, wie diese Merkmale es dem Benutzer ermöglichen, neuronale Modelle einfacher und schneller zu entwickeln und zu konstruieren sowie maßgeschneiderte Erweiterungen des PCSIM-Frameworks einfach hinzuzufügen. Des weiteren werden die Vorzüge der Integration von PCSIM in Python erläutert.

Schlüsselwörter: Probabilistische Inferenz, graphische Modelle, spikende Neuronen, Netzwerkmotive, dendritische Verarbeitung, belohnungsmoduliertes Lernen, STDP, Biofeedback, PCSIM, neuronaler Simulator, parallele Simulation, Python

Acknowledgements

First of all I would like to thank my supervisor Prof. Wolfgang Maass for giving me the opportunity to work on very exciting research topics and providing valuable guidance and continuous support throughout my PhD studies. His broad expertise, inspiring ideas and demand for excellence have greatly influenced my work. I also want to express my gratitude to Thomas Natschläger for the fruitful collaboration we had during the development of the PCSIM simulator, making it a successful and enjoyable project. Also I am very thankful to my co-authors and colleagues Robert Legenstein and Lars Büsing for contributing worthwhile ideas and doing valuable work within the research that lead to our joint publications. I would also like to thank Prof. Gordon Pipa for accepting to be the second reviewer of my thesis.

Many thanks also go to my colleagues at the Institute of Theoretical Computer Science (IGI), who with their enthusiasm for science, opennes to enroll in interesting discussions and willingness to provide me with assistance when needed, made IGI an exciting, motivating and friendly working environment. I am also grateful to Daniela Potzinger and Oliver Friedl for their assistance and support regarding administrative and hardware/software matters.

Apart from my colleagues, during my stay in Graz I got to know many exceptional people outside of work, whom I became good friends with. Thank you guys for making these years in Graz a great experience.

Finally, I would like to express my deepest gratitude to my parents Aleksandar and Ljubica for their encouragement and support in many ways throughout my life. I would also like to deeply thank my sister Ivana, for being above all always a great friend and for her enormous support during my PhD studies.

Contents

Chapter 1

Introduction

Arguably, one of the most alluring unanswered questions in modern science is how the human brain works and gives rise to high-level mental processes and behavior. There is a little doubt that any significant progress towards answering this question will have a profound impact on society, science and technology. However, the degree of difficulty of this open problem becomes evident as soon as we begin to consider some known facts about the brain. First, it has a highly complex structure: it is composed of a large number of units, approximately 100 billion neural cells and about 1000 trillion synaptic connections between them which through their synergistic activity yield higher cognitive processing. Second, it has an extremely complex function: it generates highly diverse behaviors when faced with various tasks and situations by engaging a combination of its cognitive processing abilities like perception, decision making, memory, language, motor control etc. Furthermore, the neural structures and their associated dynamical and computational processes are organized on different spatial and temporal scales that span several orders of magnitude (Churchland et al., 1993). On the spatial scale at the lowest level are the molecular and electrophysiological processes within individual neurons and synapses, then local networks of neurons or neural circuits, brain areas, systems and the whole brain at the highest level. On the temporal scale, the temporal processes range from fast stochastic dynamics of ion channels and generation of action potentials and postsynaptic responses on the order of milliseconds, to long-term synaptic plasticity mechanisms (believed to underlie long-term memory formation and learning) and developmental changes that could span hours, days or longer.

In spite of being a tremendous challenge, the scientific quest of understanding the brain has led to notable progress in the last decades. For example, at the level of individual neurons many processes related to how neurons transmit and process electrical and chemical signals are well understood. Also, with the advancement of the electrophysiology recording techniques in vivo, numerous studies have been pursued that reveal what information the spiking activity of single or small groups of neurons in different parts of the brain contains about a given stimuli, movement or specific behavior of the animal, as well as how the neurons encode this information. In another line of research, on the level of brain areas, functional neuroimaging techniques (fMRI, PET etc.) have provided means to look which parts of the brain are activated above average when performing different cognitive tasks and based on that to map higher cognitive functions to different brain regions. These are only a few of many examples of progress that has been made in different subfields of neuroscience. Still, at the level that should provide a link between the activity of individual neurons and the function of different brain areas, the level of local neural circuits at spatial scale on the order of millimeters in the cortex, many fundamental questions remain largely unanswered. Namely, it is not known what is the computational function of local networks of biological neurons, how is the computation organized, how the neural circuits self-organize in specific structures that implement the computational function, how they represent information, how they adapt and learn at the micro-level in order to support the observed learning, memory and improvement of performance at the behavioral level, and so on. In summary, we can tentatively frame these questions in three overlapping topics: the computational function of neural circuits, their realization of learning and their development.

Before one starts to analyse what the computational function of local neural circuits might be, one question that arises and is important to point out concerns the degree of uniformity of the computational algorithms across areas in the cortex: whether all neural circuits implement a specific adapted instance of a generic computational algorithm or the specifics are so large that we can not classify them as doing the same type of computation. Although currently there is not a definite answer on that, there are some facts that go strongly in favor of the generic cortical algorithm hypothesis. One frequently given argument is that, as neuroanatomists have observed, there is a striking similarity in the anatomical characteristics of neural circuits across different areas of the cortex, e.g. its laminar structure, characteristic connectivity patterns between cell types etc., and this anatomical uniformity suggests also an existent uniformity at a functional level (Douglas and Martin, 2004a). Additionally, the presence of topographic maps of sensory information in different sensory cortices, visual, auditory and somatosensory, indicates a general principle of spatial organization of information representation and processing in cortex. In fact it has been also shown that if the optic nerve of an animal (a ferret) is rerouted to the auditory cortex early in development, auditory cortex develops a retinotopic map organization normally found in visual cortex (von Melchner et al., 2000). Finally, if we treat the question from an evolutionary perspective, it is likely to assume that after evolution found a brain structure in early mammals that proved very effective for providing certain advantageous behavioral capabilities, it started replicating this structure in descendant species producing larger cortex, since it led to animals with more complex and flexible behavior and increased their chance of survival.

Specific cognitive functions presumably involve diverse information processing that operate on inputs with different dynamics and statistics. Departing from the premise that cortical computations share the same principles of organization and have similar characteristics, they have to be general enough to achieve the required diverse types of input-output mappings. Further, their inherent learning processes should be robust and powerful enough to be able to realize the required computations and be independent of the statistical properties of the processed inputs. These themes of generality of computational processes, their efficiency, robustness as well as learning capabilities are subjects of investigations in computer science and, regarding learning issues in particular, its branch machine learning. Hence, computer science is an indispensible and fruitful source of theoretical tools, models and computational frameworks actively used in tackling the question of the computational function of neural circuits.

The formulated mathematical models that capture the dynamics of parts of the brain, e.g. the stochastic dynamics of ion channels, the input-output behavior of a single neuron or the average population activity of a patch of the cortex, given usually in the form of coupled nonlinear differential equations, are in most of the cases not amenable to analytical analysis, especially in models with a high-dimensional state. Thus, numerical simulations of the created models on a computer system are integral part of every study of the computational properties of biological networks of neurons. This implies an ongoing necessity to improve the process of simulation-based analysis in all aspects and stimulates the research on techniques and algorithms for simulation of neural systems, as well as software development of neural simulation tools that implement those techniques. An important component of these efforts, in addition to research on efficient numerical integration algorithms and efficient and flexible algorithms for construction of neural models, are innovations in software design. The goal in these innovations is to create a general simulation object-oriented software framework that has an easy to use interface, has already implemented a wide range of neurobiological model components and different simulation strategies and perhaps most importantly, allows for easy user extensions on many levels. Also, as larger computing resources are available in the form of commodity clusters or supercomputer systems, one other desirable feature provided by many neural simulators is the possibility of harnessing all available computing power for simulation of larger neural network models by using distributed simulation of one large neural network on many machines.

There are two complementary approaches that are applied in studies conducting research on the computational properties and organization of computation in neural circuits: the bottom-up and the top-down approach. In the bottom-up approach, first mathematical models that describe neurobiological structures, mechanisms and processes are derived based on sufficient amount of experimental data that characterize well the studied phenomena. Then the resulting mathematical models are simulated numerically on a computer and analysed from a computational perspective where it is examined what are the computational consequences of these phenomena, i.e. what are the type of computations that they can support or carry out. Within these studues it is often analysed what is the set of input-output functions that a model can realize or learn, the way information is encoded within the model, what is the efficiency of the computation, noise robustness, possibility of scaling up etc. In the top-down approach, first a computational framework or algorithm is postulated as a possible candidate being able to explain the computations carried out in neural circuits, and then a neural circuit model is constructed that can carry out the postulated computations which at the same time is constrained by the available experimental data. Following a top-down approach in creating a model is instrumental and necessary because very often there is not enough experimental data about the structure and dynamics of neural circuits needed to constrain and build the models. Thus, the postulated computational theory can provide hypotheses about the unknown mechanisms and their biophysical or neural implementation which can be used to complete the model construction. Furthermore, the theory-influenced shaping of the models generates specific predictions which can be a valuable input for ideas about new experimental studies that can test the predictions directly or test consequences of them.

In this thesis both top-down and bottom-up methods are applied in the study of two central questions regarding the function of neural circuits: what type of computations they implement and how learning on a synaptic level yields useful computational functions on a circuit and behavioral level. Motivated by the need to best support the neural modeling and simulation requirements of the research done in this thesis and other related work, a novel software framework for neural simulations with many useful features named PCSIM was developed, which is another additional contribution of this thesis. PCSIM was successfully used in the extensive simulations in the studies in this thesis as well as in many other research projects.

Probabilistic inference in graphical models has been often proposed as a suitable candidate for explaining the computations that the brain carries out in the face of great amount of uncertainty present in the sensory inputs and its internal representations of the world. But the neural basis of these computations, i.e. how networks of spiking neurons could implement probabilistic inference, remains unclear. In chapter 2 this problem is approached, and building on previous results in (Büsing et al., 2011) several different possible implementations of probabilistic inference in graphical models with networks of spiking neurons are presented. The developed neural implementations perform probabilistic inference through Markov chain Monte Carlo sampling and use specific network structures or dendritic computations in biologically realistic neurons as basic building blocks to realize the required nonlinear computational operations. Hence, they propose that the computational function of local network structures as well as the dendritic computations in single neurons is to support the probabilistic inference operations on a larger network level. The models further suggest that the stochastic properties of biological neurons have a useful purpose to provide the necessary stochasticity in the sampling algorithm and should not be viewed as undesirable noise. The performance and scalability of the neural implementations are demonstrated through computer simulations where they have been applied on several example graphical models.

In chapter 3 it is analysed theoretically and through computer simulations what computations can be learned with reward-modulated spike-timing-dependent plasticity, a synaptic plasticity learning rule based on experimental findings about long-term synaptic efficacy changes dependent on spike times and the gating effect of neuromodulators on this type of plasticity. Spike-timing-dependent plasticity (STDP) is an experimentally observed effect about changes in synaptic efficacy that is believed to underlie the learning and long-term memory processes in the brain. Modulation of STDP with a neuromodulatory signal (e.g. dopamine) related to reward is a candidate mechanism that could explain how local synaptic changes on a micro-scale support adaptive behavioral changes based on reinforcements on a macro-scale. In chapter 3 it is shown that this plasticity rule enables spiking neurons to learn classification of temporal spike patterns, and respond with a high firing rate to one of the patterns while remaining silent for the other. It is also shown that neurons can learn with reward-modulated STDP a specific mapping from input spike patterns to output spike patterns. Moreover, it is analysed theoretically under which conditions and parameter values for the learning rule and the neuron model the learning in these learning tasks is successful. Additionally, it is demonstrated that reward-modulated STDP can explain experimental results of biofeedback learning in monkeys (Fetz and Baker, 1973) and be used to train spiking neurons to read out information from a preprocessing neural circuit. The results also suggest a functional role for spontaneous activity as performing random exploration needed in reward-based learning.

Chapter 4 gives an overview of the *P*arallel neural *C*ircuit *SIM*ulator (PCSIM) with a focus on its integration with the Python programming language. PCSIM is a neural simulation environment intended for simulation of spiking and analog neural networks with a support for distributed simulation of large-scale neural networks on multiple machines. It is implemented in C++ with its user interface exposed in the Python programming language. In chapter 4 key features of PCSIM's modular and extensible object-oriented framework and user interface are outlined and it is described how these features enable the user to develop and construct neural models easier and faster, to speed up the simulations of the models, and to add easily custom extensions to the framework in order to adapt the simulator to his own modeling needs. Further, some of the many benefits the integration of PCSIM with Python brings to the user are elucidated: high-level, easy to use, scripting interface for specification of the models, extending PCSIM with add-ons implemented in Python fostering a hybrid approach to modelling, and combined usage of PCSIM with many other scientific computing Python software packages (general or neuroscience specific). Also, the supplementary PCSIM packages implemented in pure Python that augment the PCSIM package bundle with additional useful functionalities are described.

Chapter 2 in this thesis is based on the paper *Probabilistic Inference in General Graphical Models through Sampling in Stochastic Networks of Spiking Neurons* by myself (DP), Lars Büsing (LB) and Wolfgang Maass (WM). The paper was submitted for publication in 2011 and is under review. The experiments in this work were concieved and designed by DP and WM. DP conducted the experiments and analysed the simulation results. The paper builds on the theory of neural sampling developed by LB and reported in (Büsing et al., 2011). DP and WM provided the additional theoretical derivations and analysis in the paper. DP and WM wrote the manuscript. LB provided valuable comments that helped to improve the manuscript.

Chapter 3 is based on the journal article *A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback* by

Robert Legenstein[1] (RL), myself[1] (DP) and Wolfgang Maass (WM) (PLoS Computational Biology 4(10): e1000180, 2008). In this article RL contributed the theoretical analysis, RL, DP and WM conceived and designed the experiments and DP conducted the experiments and analysed the simulation results. RL, DP and WM wrote the manuscript.

Chapter 4 is based on the journal article *PCSIM: A Parallel Simulation Environment for Neural Circuits Fully Integrated with Python* by myself (DP), Thomas Natschläger (TN) and Klaus Schuch (KS) (*Frontiers in Neuroinformatics* 3:11, 2009). The PCSIM software described in the article was developed by DP and TN, with contributions from KS. TN supervised the software development project. DP implemented and performed the computer simulation tests reported in the article. The article was written by DP and TN. KS wrote the section that describes the PYLSM package and gave useful comments for improving the manuscript.

[1]These authors contributed equally to the work in this paper.

Chapter 2

Probabilistic Inference in General Graphical Models through Sampling in Stochastic Networks of Spiking Neurons

Contents

An important open problem of computational neuroscience is the generic organization of computations in networks of neurons in the brain. It has been argued that traditional models for universal computation, such as Turing machines, are less suitable as a conceptual framework. Probabilistic inference in graphical models has been proposed as an alternative, that would be better suited for solving the computational tasks which the brain has to carry out, where decisions have to be made based on large numbers of uncertain percepts and memories. But it has remained an open problem how such computations could be carried out by networks of spiking neurons. We show here that inherent stochastic features of spiking neurons, in combination with simple nonlinear computational operations in specific network motifs and dendritic arbors, enable networks of spiking neurons to carry out probabilistic inference through sampling in general graphical models. In particular, it enables them to carry out probabilistic inference in Bayesian networks with converging arrows ("explaining away") and with undirected loops, that occur in many real-world tasks. The resulting computational model suggests that ubiquitous stochastic features of networks of spiking neurons, such as trial-to-trial variability and spontaneous activity, should not be viewed as effects of noise in deterministic computations, but rather as necessary ingredients of this underlying computational organization. We demonstrate through computer simulations that this approach can be scaled up to neural emulations of probabilistic inference in fairly large graphical models, yielding some of the most complex computations that have been carried out so far in networks of spiking neurons.

2.1 Introduction

In spite of intense theoretical and experimental research that spans almost a century, the fundamental questions how information processing in the brain is organized, and which concrete computational operations are carried out by stereotypical cortical microcircuits, have remained unanswered. Turing machines, logical inference machines, and related universal computational models would provide sufficient computational power for deterministic computations, but their structure and dynamics is incompatible with basic aspects of neural circuits. Attractor neural networks and synfire chains fair better from this perspective, but it seems difficult to implement very complex computations with them.

Probabilistic inference in Bayesian networks (Pearl, 1988) and other graphical models (Bishop, 2007; Koller and Friedman, 2009) has emerged as an alternative computational framework that is especially suited for computational tasks that the brain has to solve: The formation of coherent interpretations of incomplete and ambiguous sensory stimuli, fast learning of new information, integration of previously acquired knowledge with new information, movement planning, reasoning and decision making in the presence of uncertainty (Rao et al., 2002; Doya et al., 2007; Fiser et al., 2010). In this approach one assumes that previously acquired knowledge (facts, rules, constraints, successful responses) is encoded in a joint distribution p over numerous random variables (RVs) $z_1, \ldots, z_K$, that represent features of sensory stimuli, aspects of internal models for the environment, environmental and behavioral context, values of carrying out particular actions in particular situations (Toussaint and Goerick, 2010), goals, etc. If the values of some of these RVs assume concrete values $\boldsymbol{e}$ (e.g. because of observations, or because a particular goal needs to be reached), the distribution of the remaining variables changes in general. A typical computation that needs to be carried out for probabilistic inference for some joint distribution $p(z_1, \ldots, z_l, z_{l+1}, \ldots, z_K)$ is the evaluation of an expression of the type

$$p(z_1|\boldsymbol{e}) = \sum_{\substack{\text{all possible values} \\ v_2, \ldots, v_l \text{ for } z_2, \ldots, z_l}} p(z_1, v_2, \ldots, v_l|\boldsymbol{e}) \quad , \tag{2.1}$$

where concrete values $\boldsymbol{e}$ (the "evidence" or “observations”) have been inserted for the RVs $z_{l+1}, \ldots, z_K$. These variables are then often called observable variables, and the others latent variables. Note that the term “evidence” is somewhat misleading, since the assignment $\boldsymbol{e}$ represents some arbitrary input to a probabilistic inference computation, without any connotation that it represents correct observations or memories. The computation of the resulting marginal distribution $p(z_1|\boldsymbol{e})$ requires a summation over all possible values $v_2, \ldots, v_l$ for the RVs $z_2, \ldots, z_l$ that are currently not of interest for this probabilistic inference. This computation is in general quite complex (in fact, it is NP-complete (Koller and Friedman, 2009)) because in the worst case exponentially in l many terms need to be evaluated and summed up.

It turned out to be surprisingly difficult to elucidate how networks of neurons in the brain could possibly implement such computations, even for rather simple probability distribution p. Most previous attempts had focused on belief propagation, i.e., on distributed deterministic arithmetical computations for the evaluation of the r.h.s. of (2.1). Such computational schemes are hard to reconcile with experimental data on the dynamics of networks of neurons in the brain, for example their stochastic aspects such as trial-to-trial variability. In addition, belief propagation is not guaranteed to work for general Bayesian networks with undirected cycles. Other modelling attempts, starting with Boltzmann machines (Ackley et al., 1985), proposed that the ubiquitous stochastic aspects of neuronal responses provide clues that the brain has chosen a completely different way for carrying out probabilistic inference: by building an internal model for the probability distribution p for which inference has to be carried out, and by drawing examples from this probability distribution p. This approach is referred to as sampling in machine learning. If one has a physical realization of p, i.e. a mechanism that draws samples $(v_1, \ldots, v_l)$ of assignments to all "free" RVs $z_1, \ldots, z_l$ in (2.1) according to the distribution $p(z_1, \ldots, z_l, \boldsymbol{e})$, one can estimate $p(z_1|\boldsymbol{e})$ by just observing how often each possible value v_1 for z_1 occurs in these samples $(v_1, \ldots, v_l)$. Similarly one can infer relationships among RVs $z_1, \ldots, z_l$, e.g. whether z_1 and z_2 are correlated by simply observing how often $v_1 = v_2$ occurs in these samples.

A very successful method for such probabilistic inference through sampling has become known in machine learning under the name Markov chain Monte Carlo (MCMC) sampling (Neal, 1993), (Andrieu et al., 2003), (Koller and Friedman, 2009). The general idea is to construct a Markov chain whose set of states is exactly the set of all possible assignments $(v_1, \ldots, v_K)$ of values to the RVs of p, and whose stationary distribution of states (see Methods) is exactly the distribution p for which one wants to carry out probabilistic inference. Under some mild conditions the relative time that the Markov chain spends in each of its states is guaranteed to converge – from any initial state – to this stationary distribution p. Hence, as soon as the Markov chain provides a good approximation of p, the sequence of states $(v_1, \ldots, v_K)$ that it enters (starting from any initial state) can be viewed as almost unbiased samples from p.

For a Boltzmann machine a standard way of sampling is Gibbs sampling. This Markov chain is reversible, i.e., stochastic transitions between states do not have a preferred direction. This sampling method works well in artificial neural networks, where the effect of each neural activity lasts for exactly one discrete time step. But it is in conflict with basic features of networks of spiking neurons, where each action potential (spike) of a neuron triggers inherent temporal processes in the neuron itself (e.g. refractory processes), and through postsynaptic potentials of specific durations in other neurons to which it is synaptically connected. These inherent temporal processes of specific durations are non-reversible, and are therefore inconsistent with the mathematical model (Gibbs sampling) that underlies probabilistic inference in Boltzmann machines. However very recently a somewhat different mathematical model (sampling in non-reversible Markov chains) has emerged as an alternative framework for probabilistic inference in neural networks, that is compatible with these basic features of the dynamics of networks of spiking neurons (Büsing et al., 2011). In this approach one relates the firing activity in a network $\mathcal{N}$ of K spiking neurons $\nu_1, \ldots, \nu_K$ to sampling from a distribution $p(z_1, \ldots, z_K)$ over binary variables $z_1, \ldots, z_K$ by setting

$$z_k(t) = 1 \quad \begin{array}{l}\text{if and only if neuron } \nu_k \text{ has fired within the preceding} \\ \text{time interval } (t-\tau, t] \text{ of length } \tau \quad ,\end{array} \tag{2.2}$$

(we restrict our attention here to binary RVs; multinomial RVs could in principle be represented by WTA circuits – see Discussion). The constant τ models the average length of the effect of a spike on the firing probability of other neurons or of the same neuron, and can be set for example to $\tau = 20$ ms. However with this definition of its internal state $(z_1(t), \ldots, z_K(t))$ the dynamics of the neural network $\mathcal{N}$ can not be modelled by a Markov chain, since knowledge of this current state does not suffice for determining the distribution of states at future time points, say at time $t + 5$ ms. This distribution requires knowledge about when exactly a neuron ν_k with $z_k(t) = 1$ had fired. Therefore auxiliary random variables $\zeta_1, \ldots, \zeta_K$ with multinomial or analog values were introduced in (Büsing et al., 2011), that keep track of when exactly in the preceding time interval of length τ a neuron ν_k had fired, and thereby restore the Markov property for a Markov chain that is defined over an enlarged state set consisting of all possible values of $z_1, \ldots, z_K$ and $\zeta_1, \ldots, \zeta_K$. However the introduction of these hidden variables $\zeta_1, \ldots, \zeta_K$, that keep track of inherent temporal processes in the network $\mathcal{N}$ of spiking neurons, comes at the price that the resulting Markov chain is no longer reversible (because these temporal processes are not reversible). But it was shown in (Büsing et al., 2011) that one can prove nevertheless for any distribution $p(z_1, \ldots, z_K)$ for which the so-called neural computability condition (NCC), see below, can be satisfied by a network $\mathcal{N}$ of spiking neurons, that $\mathcal{N}$ defines a non-reversible Markov chain whose stationary distribution is an expanded distribution $p(z_1, \ldots, z_K, \zeta_1, \ldots, \zeta_K)$, whose marginal distribution over $z_1, \ldots, z_K$ (which results when one ignores the values of the hidden variables $\zeta_1, \ldots, \zeta_K$) is the desired distribution $p(z_1, \ldots, z_K)$. Hence a network $\mathcal{N}$ of spiking neurons can sample from any distribution $p(z_1, \ldots, z_K)$ for which the NCC can be satisfied. This implies that any neural system that contains such network $\mathcal{N}$ can carry out the probabilistic inference task (2.1): The evidence $\boldsymbol{e}$ could be implemented through external inputs that force neuron ν_k to fire at a high rate if $z_k = 1$ in $\boldsymbol{e}$, and not to fire if $z_k = 0$ in $\boldsymbol{e}$. In order to estimate $p(z_1|\boldsymbol{e})$, it suffices that some readout neuron estimates (after some initial transient phase) the resulting firing rate of the neuron ν_1 that represents RV z_1.

The NCC requires that for each RV z_k the firing probability density $\rho_k(t)$ of some corresponding neuron ν_k at time t satisfies, if the neuron is not in a refractory period

$$\rho_k(t) = \frac{1}{\tau} \cdot \frac{p(z_k = 1|\boldsymbol{z}_{\backslash k})}{p(z_k = 0|\boldsymbol{z}_{\backslash k})} \quad , \tag{2.3}$$

where $\mathbf{z}_{\backslash k}$ denotes the current value of all other RVs, i.e., all z_i with $i \neq k$. We use in this work the same model for a stochastic neuron as in (Büsing et al., 2011) (continuous time case), which can be matched quite well to biological data according to (Jolivet et al., 2006). In the simpler version of this neuron model one assumes that it has an absolute refractory period of length τ, and that the instantaneous firing probability $\rho_k(t)$ satisfies outside of its refractory period $\rho_k(t) = \frac{1}{\tau}\exp(u_k(t))$, where $u_k(t)$ is its membrane potential (see Methods for an account of the more complex neuron model with a relative refractory period from (Büsing et al., 2011), that we have also tested in our simulations). The NCC from (2.3) can then be reformulated as a condition on the membrane potential of the neuron

$$u_k(t) = \log \frac{p(z_k = 1|\boldsymbol{z}_{\backslash k})}{p(z_k = 0|\boldsymbol{z}_{\backslash k})} \quad . \tag{2.4}$$

Let us consider a Boltzmann distribution p of the form

$$p(z_1, \ldots, z_K) = \frac{1}{Z} \exp\left(\sum_{i,j} \frac{1}{2} W_{ij} z_i z_j + \sum_i b_i z_i\right) \tag{2.5}$$

with symmetric weights (i.e., $W_{ij} = W_{ji}$) that vanish on the diagonal (i.e., $W_{ii} = 0$). In this case the NCC can be satisfied by a $u_k(t)$ that is *linear* in the postsynaptic potentials that neuron ν_k receives from the neurons ν_i that represent other RVs z_i:

$$u_k(t) = b_k + \sum_{i=1}^{K} W_{ki}\; z_i(t) \quad , \tag{2.6}$$

where b_k is the bias of neuron ν_k (which regulates its excitability), W_{ki} is the strength of the synaptic connection from neuron ν_i to ν_k, and $z_i(t)$ approximates the time course of the postsynaptic potential caused by a firing of neuron ν_i at some time $t_i^f < t$ ($z_i(t)$ assumes value 1 during the time interval $[t_i^f, t_i^f + \tau)$, otherwise it has value 0).

However, it is well known that probabilistic inference for distributions of the form (2.5) is too weak to model various important computational tasks that the brain is obviously able to solve, at least without auxiliary variables. While (2.5) only allows pairwise interactions between RVs, numerous real world probabilistic inference tasks require inference for distributions with higher order terms. For example, it has been shown that human visual perception involves “explaining away”, a well known effect in probabilistic inference, where a change in the probability of one competing hypothesis for explaining some observation affects the probability of another competing hypothesis (Kersten and Yuille, 2003). Such effects can usually only be captured with terms of order at least 3, since 3 RVs (for 2 hypotheses and

1 observation) may interact in complex ways. A well known example from visual perception is shown in Fig. 2.1, for a probability distribution p over 4 RVs $z_1, \dots, z_4$, where z_1 is defined by the perceived relative reflectance of two abutting 2D areas, z_2 by the perceived 3D shape of the observed object, z_3 by the observed shading of the object, and z_4 by the contour of the 2D image. The difference in shading of the two abutting surfaces in Fig. 2.1A could be explained either by a difference in reflectance of the two surfaces, or by an underlying curved 3D shape. The two different contours (RV z_4) in the upper and lower part of Fig. 2.1A influence the likelihood of a curved 3D shape (RV z_3). In particular, a perceived curved 3D shape "explains away" the difference in shading, thereby making an uniform reflectance more likely. The results of (Knill and Kersten, 1991) and numerous related results suggest that the brain is able to carry out probabilistic inference for more complex distributions than the 2^{nd} order Boltzmann distribution (2.5).

We show in this work that the neural sampling method of (Büsing et al., 2011) can be extended to any probability distribution p, in particular to distributions with higher order dependencies among RVs, by using auxiliary spiking neurons in $\mathcal{N}$ that do not directly represent RVs z_k, or by using nonlinear computational processes in multi-compartment neuron models. As one can expect, the number of required auxiliary neurons or dendritic branches increases with the complexity of the probability distribution p for which the resulting network of spiking neurons has to carry out probabilistic inference. Various types of graphical models (Koller and Friedman, 2009) have emerged as convenient frameworks for characterizing the complexity of distributions p from the perspective of probabilistic inference for p.

We will focus in this work on Bayesian networks, a common type of graphical model for probability distributions. But our results can also be applied for other types of graphical models. A Bayesian network is a directed graph (without directed cycles), whose nodes represent RVs $z_1, \dots, z_K$. Its graph structure indicates that $p(z_1, \dots, z_K)$ admits a factorization of the form

$$p(z_1, \dots, z_k) = \prod_{k=1}^{K} p(z_k|\mathrm{pa}(z_k)), \tag{2.7}$$

where $\mathrm{pa}(z_k)$ is the set of all (direct) parents of the node indexed by z_k (see Fig. 2.1B, 2.7A, 2.9 for examples). For example, the Bayesian network in Fig. 2.1B implies that the factorization $p(z_1, z_2, z_3, z_4) = p(z_1)p(z_2)p(z_3|z_1, z_2)p(z_4|z_3)$ is possible.

We show that the complexity of the resulting network of spiking neurons for carrying out probabilistic inference for p can be bounded in terms of the graph complexity of the Bayesian network that gives rise to the factorization (2.7). More precisely, we present three different approaches for constructing such networks of spiking neurons:

- through a reduction of p to a Boltzmann distribution (2.5) with auxiliary RVs
- through a Markov blanket expansion of the r.h.s. of the NCC (2.4)

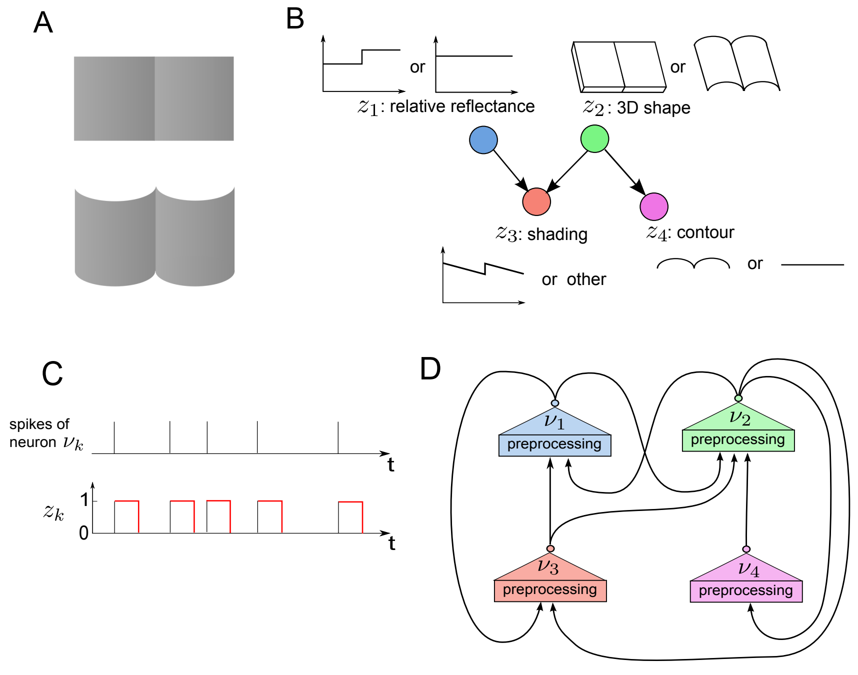

Figure 2.1: See next page for figure caption.

- through a factorized expansion of the r.h.s. of the NCC (2.4)

We will show that there exist two different neural implementation options for each of the last two approaches, using either specific network motifs or dendritic processing for nonlinear computation steps. This yields altogether 5 different options for emulating probabilistic inference in Bayesian networks through sampling via the inherent stochastic dynamics of networks of spiking neurons. Furthermore we will exhibit characteristic differences in the complexity and performance of the resulting networks, and relate these to the complexity of the underlying Bayesian network. But in contrast to some previously suggested emulations of probabilistic inference by networks of spiking neurons, all 5 of these neural implementation options can readily be applied to Bayesian networks where several arcs converge to a node (giving rise to the "explaining away" effect), and to Bayesian networks with undirected cycles ("loops"). All methods for probabilistic inference from general graphical models that we propose in this work are from the mathematical perspective special cases of MCMC sampling. However in view of the fact that they expand the neural sampling approach of (Büsing et al., 2011), we will refer to them more specifically as neural sampling.

We show through computer simulations for three different Bayesian networks of different sizes that neural sampling can be carried quite fast with the help of the second and third approach, providing good inference results within a behaviorally relevant time span of a few hundred ms. One of these Bayesian networks addresses the previously described classical "explaining away" effect in visual perception from

Figure 2.1: The visual perception experiment of (Knill and Kersten, 1991) that demonstrates "explaining away", and the Bayesian network that models the phenomenon. **A)** Two visual stimuli, each exhibiting the same luminance profile in the horizontal direction. The two visual stimuli differ only with regard to their contours, which suggest different 3D shapes (flat versus 2 cylinders). This in turn influences our perception of the reflectance of the two halves of each stimulus (a step in the reflectance at the middle line, versus uniform reflectance): the cylindrical 3D shape "explains away" the reflectance step. **B)** The Bayesian network that models this effect consists of 4 RVs z_1, z_2, z_3, and z_4. The relative reflectance (z_1) of the two halfs can have two values, different ($z_1 = 1$) or the same ($z_1 = 0$). The perceived 3D shape is either cylindrical ($z_2 = 1$) or flat ($z_2 = 0$). The relative reflectance and the 3D shape are direct causes of the shading (luminance change) of the surfaces denoted as z_3, which can have the profile like in panel A ($z_3 = 1$) or a different one ($z_3 = 0$). The 3D shape of the object causes different perceived contours z_4, which can be either straight ($z_4 = 0$) or curved ($z_4 = 1$). The observed variables are the contour (z_4) and the shading (z_3) of the stimulus. Subjects infer the value of the relative reflectance and the 3D shape based on this evidence. The probability distribution $p(z_1, z_2, z_3, z_4)$ of the Bayesian network factorizes to $p(z_1)p(z_2)p(z_3|z_1, z_2)p(z_4|z_2)$, and the inference problem is to calculate the marginal posterior probability distributions $p(z_1|z_3, z_4)$ and $p(z_2|z_3, z_4)$. **C)** Each of these RVs z_k are represented in our neural emulations of probabilistic inference by a principal neuron ν_k in such a way, that each spike of ν_k sets the RV z_k to 1 for a time period of length τ. **D)** The structure of a network of spiking neurons that performs probabilistic inference for the Bayesian network of panel B through sampling from conditionals of the underlying joint probability distribution $p(z_1, z_2, z_3, z_4)$. Each principal neuron employs preprocessing to satisfy the neural computability condition (NCC), either by dendritic processing or by a preprocessing circuit. Note that, in contrast to the directed acyclic Bayesian network of panel B, this computational network (see Fig. 2.6 for a concrete neural emulation) is recurrently connected, resulting from the fact that during probabilistic inference information flows also against the direction of the arcs in the Bayesian network (an example is the "explaining away" effect).

Fig. 2.1. The other two Bayesian networks not only contain numerous "explaining away" effects, but also undirected cycles. Altogether, our computer simulations and our theoretical analysis demonstrate that networks of spiking neurons can emulate probabilistic inference for general Bayesian networks. Hence we propose to view probabilistic inference in graphical models as a generic computational paradigm, that can help us to understand the computational organization of networks of neurons in the brain, and in particular the computational role of precisely structured cortical microcircuit motifs.

2.2 Results

We present several ways how probabilistic inference for a given joint distribution $p(z_1, \ldots, z_K)$, that is not required to have the form of a 2^{nd} order Boltzmann distribution (2.5), can be carried out through sampling from the inherent dynamics of a recurrent network $\mathcal{N}$ of stochastically spiking neurons. All these approaches are based on the idea that such network $\mathcal{N}$ of spiking neurons can be viewed – for a suitable choice of its architecture and parameters – as a "physical model" for the distribution p, in the sense that its distribution of network states converges to p, from any initial state. Then probabilistic inference for p can be easily carried out by any readout neuron that observes the resulting network states, or the spikes from one or several neurons in the network. This holds not only for sampling from the prior distribution p, but also for sampling from the posterior after some evidence $\boldsymbol{e}$ has become available (see (2.1)). The link between network states of $\mathcal{N}$ and the RVs $z_1, \ldots, z_K$ is provided by assuming that there exists for each RV z_k a neuron ν_k so that each time when ν_k fires, it sets the associated binary RV z_k to 1 for a time period of some length τ (see Fig. 2.1C). We refer to neurons ν_k that represent in this way a RV z_k as principal neurons. All other neurons are referred to as auxiliary neurons.

The mathematical basis for analyzing the distribution of network states, and relating it to a given distribution p, is provided by the theory of Markov chains. More precisely, it was shown in (Büsing et al., 2011) that by introducing for each principal neuron ν_k an additional hidden analog RV ζ_k, that keeps track of time within the time interval of length τ after a spike of ν_k, one can model the dynamics of the network $\mathcal{N}$ by a non-reversible Markov chain. This Markov chain is non-reversible, in contrast to Gibbs sampling or other Markov chains that are usually considered in Machine Learning and in the theory of Boltzmann machines, because this facilitates the modelling of the temporal dynamics of spiking neurons, in particular refractory processes within a spiking neuron after a spike and temporally extended effects of its spike on the membrane potential of other neurons to which it is synaptically connected (postsynaptic potentials). The underlying mathematical theory guarantees that nevertheless the distribution of network states of this Markov chain converges (for the "original" RVs z_k) to the given distribution p, provided that the NCC (2.4) is met. This theoretical result reduces our goal, to demonstrate ways how a network of spiking neurons can carry out probabilistic inference in general graphical models, to the analysis of possibilities for satisfying the NCC (2.4) in networks of spiking neurons. The networks of spiking neurons that we construct and analyze build primarily on the model for neural sampling in continuous time from (Büsing et al., 2011), since this is the more satisfactory model from the biological perspective. But all our results also hold for the mathematically simpler version with discrete time.

We exhibit both methods for satisfying the NCC with the help of auxiliary neurons in networks of point neurons, and in networks of multi-compartment neuron models (where no auxiliary neurons are required). All neuron models that we consider are stochastic, where the probability density function for the firing of a neuron at time t (provided it is currently not in a refractory state) is proportional to $\exp(u(t))$, where $u(t)$ is its current membrane potential at the soma. We assume (as in (Büsing et al., 2011)) that in a point neuron model the membrane potential $u(t)$ can be written as a linear combination of postsynaptic potentials. Thus if the principal neuron ν_k is modelled as a point neuron, we have

$$u_k(t) = b_k + \sum_{i=1}^{K} W_{ki}\ z_i(t) \quad , \tag{2.8}$$

where b_k is the bias of neuron ν_k (which regulates its excitability), W_{ki} is the strength of the synaptic connection from neuron ν_i to ν_k, and $z_i(t)$ approximates the time course of the postsynaptic potential in neuron ν_k caused by a firing of neuron ν_i. The ideal neuron model from the perspective of the theory of (Büsing et al., 2011) has an absolute refractory period of length τ, which is also the assumed length of a postsynaptic potential (EPSP or IPSP). But it was shown there through computer simulations that neural sampling can be carried out also with stochastically firing neurons that have a relative refractory period, i.e. the neuron can fire with some probability with an interspike interval of less than τ. In addition, it was shown there theoretically that the resulting neural network samples from a slight variation of the target distribution p, that is in most cases practically indistinguishable.

Before we describe two different theoretical approaches for satisfying the NCC, we first consider an even simpler method for extending the neural sampling approach from (Büsing et al., 2011) to arbitrary distributions p: through a reduction to 2^{nd} order Boltzmann distributions (2.5) with auxiliary RVs.

2.2.1 Second Order Boltzmann Distributions with Auxiliary Random Variables (Implementation 1)

It is well known (Ackley et al., 1985) that any probability distribution $p(z_1, \ldots, z_K)$, with arbitrarily large factors in a factorization such as (2.7), can be represented as marginal distribution

$$p(\mathbf{z}) = \sum_{\mathbf{x} \in \mathbf{X}} p(\mathbf{z}, \mathbf{x}) \tag{2.9}$$

of an extended distribution $p(\mathbf{z}, \mathbf{x})$ with auxiliary random variables $\mathbf{x}$, that can be factorized into factors of degrees at most 2. This can be seen as follows. Let $p(\mathbf{z})$ be an arbitrary probability distribution over binary variables with higher-order factors $\phi_c(\mathbf{z}^c)$. Thus

$$p(\mathbf{z}) = \frac{1}{Z} \prod_{c=1}^{C} \phi_c(\mathbf{z}^c) \quad , \tag{2.10}$$

where $\mathbf{z}^c$ is a vector composed of the RVs that the factor ϕ_c depends on and Z is a normalization constant. We additionally assume that $p(\mathbf{z})$ is non-zero for each value of $\mathbf{z}$. The simple idea is to introduce for each possible assignment $\mathbf{v}$ to the RVs $\mathbf{z}^c$ in a higher order factor $\phi_c(\mathbf{z}^c)$ a new RV $x^c_{\mathbf{v}}$, that has value 1 only if $\mathbf{v}$ is the current assignment of values to the RVs in $\mathbf{z}^c$. We will illustrate this idea through the concrete example of Fig. 2.1. Since there is only one factor that contains more than 2 RVs in the probability distribution of this example (see caption of Fig. 2.1C), the conditional probability $p(z_3|z_1, z_2)$, there will be 8 auxiliary RVs $x_{\mathbf{000}}$, $x_{\mathbf{001}}$,

..., $x_{\mathbf{111}}$ for this factor, one for each of the 8 possible assignments to the 3 RVs in $p(z_3|z_1, z_2)$. Let us consider a particular auxiliary RV, e.g. $x_{\mathbf{001}}$. It assumes value 1 only if $z_1 = 0$, $z_2 = 0$, and $z_3 = 1$. This constraint for $x_{\mathbf{001}}$ can be enforced through second order factors between $x_{\mathbf{001}}$ and each of the RVs z_1, z_2 and z_3. For example, the second order factor that relates $x_{\mathbf{001}}$ and z_1 has a value of 0 if $x_{\mathbf{001}} = 1$ and $z_1 = 1$ (i.e., if z_1 is not compatible with the assignment 001), and value 1 otherwise. The individual values of the factor $p(z_3|z_1, z_2)$ for different assignments to z_1, z_2 and z_3 are introduced in the extended distribution $p(\mathbf{z}, \mathbf{x})$ through first-order factors, one for each auxiliary RV $x^c_{\mathbf{v}}$. Specifically, the first-order factor that depends on $x_{\mathbf{001}}$ has value $\mu p(z_3 = 1|z_1 = 0, z_2 = 0) - 1$ (where μ is a constant that rescales the values of the factors so that $\mu p(z_3|z_1, z_2) > 1$ for all assignments to z_1, z_2 and z_3) if $x_{\mathbf{001}} = 1$, and value 1 otherwise. Further details of the construction method for $p(\mathbf{z}, \mathbf{x})$ are given in the Methods section, together with a proof of (2.9).

The resulting extended probability distribution $p(\mathbf{z}, \mathbf{x})$ has the property that, in spite of deterministic dependencies between the RVs $\mathbf{z}$ and $\mathbf{x}$, the state set of the resulting Markov chain (that consists of all non-forbidden value assignments to $\mathbf{z}$ and $\mathbf{x}$) is connected. In the previous example a non-forbidden value assignment is $x_{\mathbf{001}} = 1$ and $z_1 = 0, z_2 = 0, z_3 = 1$. But $x_{\mathbf{001}} = 0, z_1 = 0, z_2 = 0, z_3 = 1$ is also a non-forbidden value assignment. Such non-forbidden value assignments to the auxiliary RVs $\mathbf{x}^c$ corresponding to one higher order factor, where all of them assume value of 0 regardless of the values of the $\mathbf{z}^c$ RVs provide transition points for paths of probability > 0 that connect any two non-forbidden value assignments (without requiring that 2 or more RVs switch their values simultaneously). The resulting connectivity of all non-forbidden states (see Methods for a proof) implies that this Markov chain, which can be realized through a network $\mathcal{N}$ of spiking neurons according to (Büsing et al., 2011), has $p(\mathbf{z}, \mathbf{x})$ as its unique stationary distribution. The given distribution $p(\mathbf{z})$ arises as marginal distribution of this stationary distribution of $\mathcal{N}$, hence one can use $\mathcal{N}$ to sample from $p(\mathbf{z})$ (just ignore the firing activity of neurons that correspond to auxiliary RVs $x^c_{\mathbf{v}}$).

Since the number of RVs in the extended probability distribution $p(\mathbf{z}, \mathbf{x})$ can be much larger than the number of RVs in $p(\mathbf{z})$, the corresponding spiking neural network samples from a much larger probability space. This, as well as the presence of deterministic relations between the auxiliary and the main RVs in the expanded probability distribution, slow down the convergence of the resulting Markov chain to its stationary distribution. We show however in the following, that there are several alternatives for sampling from an arbitrary distribution $p(\mathbf{z})$ through a network of spiking neurons. These alternative methods do not introduce auxiliary RVs $\mathbf{x}$, but rather aim at directly satisfying the NCC (2.4) in a network of spiking neurons. In Computer Simulation I (see Fig. 2.3) we have compared the resulting convergence speed with that of the previously described method via auxiliary RVs. It turns out that the alternative strategy provides an about 10 fold speed-up for the Bayesian network of Fig. 2.1B.

2.2.2 Using the Markov Blanket Expansion of the Log-odd Ratio

Assume that the distribution p for which we want to carry out probabilistic inference is given by some arbitrary Bayesian network B. There are two different options for satisfying the NCC for p, which differ in the way by which the term on the r.h.s. of the NCC (2.4) is expanded. The option that we will analyze first uses from the structure of the Bayesian network B only the information about which RVs are in the Markov blanket of each RV z_k. The Markov blanket B_k of the corresponding node z_k in B (which consists of the parents, children and coparents of this node) has the property that z_k is independent from all other RVs once any assignment $\mathbf{v}$ of values to the RVs $\mathbf{z}^{B_k}$ in the Markov blanket has been fixed. Hence $p(z_k|\mathbf{z}_{\backslash k}) = p(z_k|\mathbf{z}^{B_k})$, and the term on the r.h.s. of the NCC (2.4) can be expanded as follows:

$$\log \frac{p(z_k = 1|\mathbf{z}^{B_k} = \mathbf{z}^{B_k}(t))}{p(z_k = 0|\mathbf{z}^{B_k} = \mathbf{z}^{B_k}(t))} = \sum_{\mathbf{v} \in Z^{B_k}} w_{\mathbf{v}}^k \cdot [\mathbf{z}^{B_k}(t) = \mathbf{v}] \quad , \tag{2.11}$$

where

$$w_{\mathbf{v}}^k = \log \frac{p(z_k = 1|\mathbf{z}^{B_k} = \mathbf{v})}{p(z_k = 0|\mathbf{z}^{B_k} = \mathbf{v})} \quad . \tag{2.12}$$

The sum indexed by $\mathbf{v}$ runs over the set Z^{B_k} of all possible assignments of values to $\mathbf{z}^{B_k}$, and $[\mathbf{z}^{B_k} = \mathbf{v}]$ denotes a predicate which has value 1 if the condition in the brackets is true, and to 0 otherwise. Hence, for satisfying the NCC it suffices if there are auxiliary neurons, or dendritic branches, for each of these $\mathbf{v}$, that become active if and only if it becomes clear from the firing activity of the principal neurons ν_i that represent the variables z_i in the Markov blanket B_k, that these variables currently assume the value $\mathbf{v}$. The corresponding term $w_{\mathbf{v}}^k$ can be implemented with the help of the bias b_k (see (2.8)) of the auxiliary neuron that corresponds to the assignment $\mathbf{v}$, resulting in a value of its membrane potential equal to the r.h.s. of the NCC (2.4). We will discuss this implementation option below as Implementation 2. In the subsequently discussed implementation option (Implementation 3) all principal neurons will be multi-compartment neurons, and no auxiliary neurons are needed. In this case $w_{\mathbf{v}}^k$ scales the amplitude of the signal from a specific dendritic branch to the soma of the multi-compartment principal neuron ν_k.

2.2.2.1 Implementation with Auxiliary Neurons (Implementation 2)

We illustrate the implementation of the Markov blanket expansion approach through auxiliary neurons for the concrete example of the RV z_1 in the Bayesian network of Fig. 2.1B (see Methods for a discussion of the general case). Its Markov blanket B_1 consists here of the RVs z_2 and z_3. Hence the resulting neural circuit (see Fig. 2.2) for satisfying the NCC for the principal neuron ν_1 uses 4 auxiliary neurons $\alpha_{00}, \alpha_{01}, \alpha_{10}$ and α_{11}, one for each of the 4 possible assignments $\mathbf{v}$ of values to the RVs z_2 and z_3. Each firing of one of these auxiliary neurons should cause an immediately subsequent firing of the principal neuron ν_1. Lateral inhibition among these auxiliary neurons can make sure that after a firing of an auxiliary neuron no

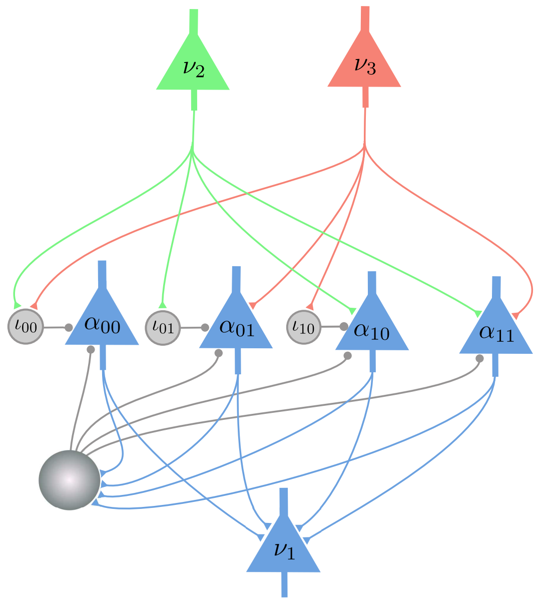

Figure 2.2: Implementation 2 (the neural implementation with auxiliary neurons, that uses the Markov blanket expansion of the log-odd ratio), for the explaining away motif of the Bayesian network from Fig. 2.1B. There are 4 auxiliary neurons, one for each possible value assignment to the RVs z_2 and z_3 in the Markov blanket of z_1. The principal neuron ν_2 (ν_3) connects to the auxiliary neuron $\alpha_{\mathbf{v}}$ directly if z_2 (z_3) has value 1 in the assignment $\mathbf{v}$, or via an inhibitory inter-neuron $\iota_{\mathbf{v}}$ if z_2 (z_3) has value 0 in $\mathbf{v}$. The auxiliary neurons connect with a strong excitatory connection to the principal neuron ν_1, and drive it to fire whenever any one of them fires. The larger gray circle represents the lateral inhibition between the auxiliary neurons .

other auxiliary neuron fires during the subsequent time interval of length τ, thereby implementing the required absolute refractory period of the theoretical model from (Büsing et al., 2011). The presynaptic principal neuron $\nu_2(\nu_3)$ is connected to the auxiliary neuron $\alpha_{\mathbf{v}}$ directly if $\mathbf{v}$ assumes that $z_2(z_3)$ has value 1, otherwise via an inhibitory interneuron $\mathbf{v}$ (see Fig. 2.2). In case of a synaptic connection via an inhibitory interneuron, a firing of $\nu_2(\nu_3)$ prevents a firing of this auxiliary neuron during the subsequent time interval of length τ. The direct excitatory synaptic connections from ν_2 and ν_3 raise the membrane potential of that auxiliary neuron $\alpha_{\mathbf{v}}$, for which $\mathbf{v}$ agrees with the current values of the RVs $z_2(t)$ and $z_3(t)$, so that it reaches the value $w^k_{\mathbf{v}}$, and fires with a probability equal to the r.h.s. of the NCC (2.4) during the time interval within which the value assignment $\mathbf{v}$ remains valid. The other 3 auxiliary neurons are during this period either inhibited by the inhibitory interneurons, or do not receive enough excitatory input from the direct connections to reach a significant firing probability. Hence, the principal neuron ν_1 will always be driven to fire just by a single auxiliary neuron $\alpha_{\mathbf{v}}$ corresponding to the current value of the variables $z_2(t)$ and $z_3(t)$, and will fire immediately after $\alpha_{\mathbf{v}}$ fires.

As $\alpha_{\mathbf{v}}$ has a firing probability that satisfies the r.h.s. of the NCC (2.4) temporally during the time interval while $z_2(t)$ and $z_3(t)$ are consistent with $\mathbf{v}$, the firing of the principal neuron ν_1 satisfies the r.h.s. of the NCC (2.4) at any moment in time.

2.2.2.2 Computer Simulation I: Comparison of two Methods for Emulating "Explaining Away" in Networks of Spiking Neurons

In our preceding theoretical analysis we have exhibited two completely different methods for emulating in networks of spiking neurons probabilistic inference in general graphical models through sampling: either by a reduction to 2^{nd} order Boltzmann distributions (2.5) through the introduction of auxiliary RVs (Implementation 1), or by satisfying the NCC (2.3) via the Markov blanket expansion. We have tested the accuracy and convergence speed of both methods for the Bayesian network of Fig. 2.1B, and the results are shown in Fig.2.3. The approach via the NCC converges substantially faster.

2.2.2.3 Implementation with Dendritic Computation (Implementation 3)

We now show that the Markov blanket expansion approach can also be implemented through dendritic branches of multi-compartment neuron models (see Methods) for the principal neurons, without using auxiliary neurons (except for inhibitory interneurons). We will illustrate the idea through the same Bayesian network example as discussed in Implementation 2, and refer to Methods for a discussion of the case of arbitrary Bayesian networks. Fig. 2.4 shows the principal neuron ν_1 in the spiking neural network for the Bayesian network of Fig. 2.1B. It has 4 dendritic branches $\delta_{00}, \delta_{01}, \delta_{10}$ and δ_{11}, each of them corresponding to one assignment $\mathbf{v}$ of values to the variables z_2 and z_3 in the Markov blanket of z_1. The input connections from the principal neurons ν_2 and ν_3 to the dendritic branches of ν_1 follow the same pattern as the connections from ν_2 and ν_3 to the auxiliary neurons in Implementation 2. Let $\mathbf{v}$ be an assignment that corresponds to the current values of the variables $z_2(t)$ and $z_3(t)$. The efficacies of the synapses at the dendritic branches and their thresholds for initiating a dendritic spike are chosen so that the total synaptic input to the dendritic branch $\delta_{\mathbf{v}}$ is then strong enough to cause a dendritic spike in the branch, that contributes to the membrane potential at the soma a component whose amplitude[1] is equal to the parameter $w^1_{\mathbf{v}}$ in (2.11). This amplitude could for example be controlled by the branch strength of this dendritic branch (see (Losonczy et al., 2008; Legenstein and Maass, 2011)). The parameters can be chosen so that all other dendritic branches do not receive enough synaptic input to reach the local threshold for initiating a dendritic spike, and therefore do not affect the membrane potential at the soma. Hence, the membrane potential at the soma of ν_1 will be equal to the

[1]Since the parameters $w^k_{\mathbf{v}}$ in (2.11) can have both positive and negative values and the amplitude of the dendritic spikes and the excitatory synaptic efficacy is a positive quantity, in this and the following neural implementations we always add a positive constant to $w^k_{\mathbf{v}}$ to shift it into the positive range. We subtract the same constant value from the steady state of the membrane potential.

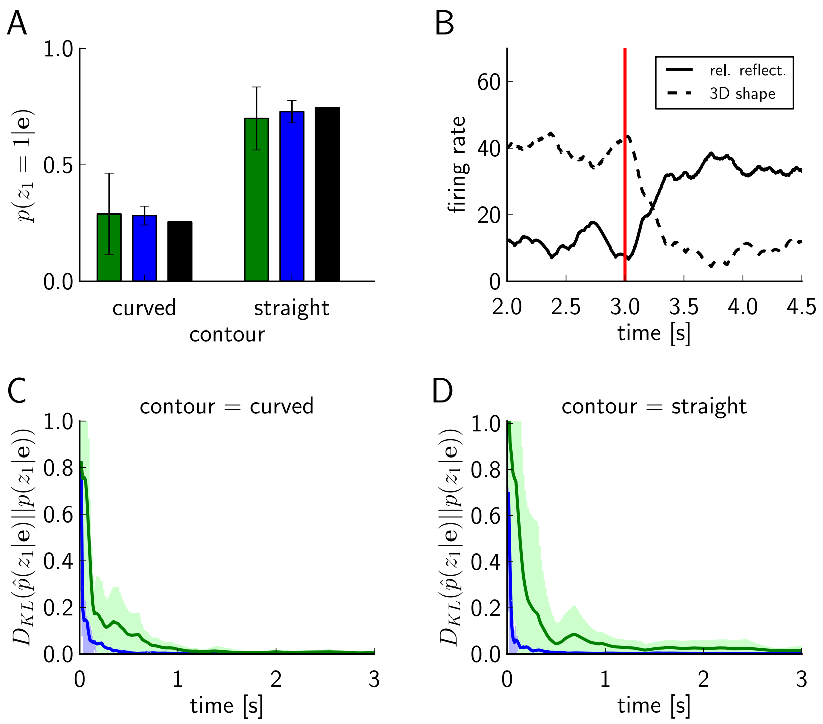

Figure 2.3: Results of Computer Simulation I: Performance comparison between an ideal version of Implementation 1 (use of auxiliary RVs, results shown in *green*) and an ideal version of implementations that satisfy the NCC (results shown in *blue*) for probabilistic inference in the Bayesian network of Fig. 2.1B ("explaining away"). Evidence $\boldsymbol{e}$ (see (2.1)) is entered for the RVs z_3 and z_4, and the marginal probability $p(z_1|\boldsymbol{e})$ is estimated. **A)** Target values of $p(z_1|\boldsymbol{e})$ for $\boldsymbol{e} = (1,1)$ and $\boldsymbol{e} = (1,0)$ are shown in black, results from sampling for 0.5 s from a network of spiking neurons are shown in green and blue. Panels **C)** and **D)** show the temporal evolution of the Kullback-Leibler divergence between the resulting estimates of $p(z_1|\boldsymbol{e})$ through neural sampling, averaged over 10 trials for $\boldsymbol{e} = (1,1)$ in C) and for $\boldsymbol{e} = (1,0)$ in D). The green and blue areas around the green and blue curves represent the unbiased value of the standard deviation. Panels A, C, D show that both approaches yield correct probabilistic inference through neural sampling, but the approach via satisfying the NCC converges about 10 times faster. **B)** The firing rates of principal neuron ν_1 (solid line) and of the principal neuron ν_2 (dashed line) in the approach via satisfying the NCC, estimated with a sliding window (alpha kernel $K(t) = \frac{t}{\tau}\ \mathtt{exp}\ (-\frac{t}{\tau}), \tau = 0.1\mathtt{s}$). In this experiment the evidence $\boldsymbol{e}$ was switched after 3 s (red vertical line) from $\boldsymbol{e} = (1,1)$ to $\boldsymbol{e} = (1,0)$. The "explaining away" effect is clearly visible from the complementary evolution of the firing rates of the neurons ν_1 and ν_2. The estimated marginal posterior is calculated for each time point from the samples (number of spikes) from the beginning of the simulation (or from $t = 3$s for the second inference query with $\boldsymbol{e} = (1,0)$).

contribution from the currently active dendritic branch $w^1_{\mathbf{v}}$, implementing thereby the r.h.s of (2.11).

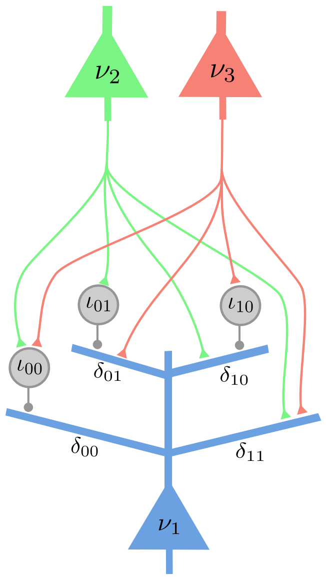

Figure 2.4: Implementation 3 (the neural implementation with dendritic computation that uses the Markov blanket expansion of the log-odd ratio), for the same explaining away motif as in Fig. 2.2. The principal neuron ν_1 has 4 dendritic branches, one for each possible assignment of values $\mathbf{v}$ to the RVs z_2 and z_3 in the Markov blanket of z_1. The dendritic branches of neuron ν_1 receive synaptic inputs from the principal neurons ν_2 and ν_3 either directly, or via an interneuron (analogously as in Fig. 2.2). It is required that at any moment in time exactly one of the dendritic branches (that one, whose index $\mathbf{v}$ agrees with the current firing states of ν_2 and ν_3) generates dendritic spikes, whose amplitude at the soma determines the current firing probability of ν_1.

2.2.3 Using the Factorized Expansion of the Log-odd Ratio

The second strategy to expand the log-odd ratio on the r.h.s. of the NCC (2.4) uses the factorized form (2.10) of the probability distribution $p(\mathbf{z})$. This form allows us to rewrite the log-odd ratio in (2.4) as a sum of log terms, one for each factor ϕ_c, $c \in C^k$, that contains the RV z_k. One can write each of these terms as a sum over all possible assignments $\mathbf{v}$ of values of the variables $\mathbf{z}^c$ the factor ϕ_c depends on (except z_k). This yields

$$\log \frac{p(z_k = 1|\mathbf{z}_{\setminus k} = \mathbf{z}_{\setminus k}(t))}{p(z_k = 0|\mathbf{z}_{\setminus k} = \mathbf{z}_{\setminus k}(t))} = \sum_{c \in C^k} \left(\sum_{\mathbf{v} \in Z^c_{\setminus k}} w^{c,k}_{\mathbf{v}} \cdot [\mathbf{z}^c_{\setminus k}(t) = \mathbf{v}] \right) \quad , \tag{2.13}$$

where $\mathbf{z}^c_{\setminus k}$ is a vector composed of the RVs $\mathbf{z}^c$ that the factor c depends on – without z_k, and $\mathbf{z}^c_{\setminus k}(t)$ is the current value of this vector at time t. $Z^c_{\setminus k}$ denotes the set of all possible assignments to the RVs $\mathbf{z}^c_{\setminus k}$. The parameters $w^{c,k}_{\mathbf{v}}$ are set to

$$w^{c,k}_{\mathbf{v}} = \log \frac{\phi_c(\mathbf{z}^c_{\setminus k} = \mathbf{v}, z_k = 1)}{\phi_c(\mathbf{z}^c_{\setminus k} = \mathbf{v}, z_k = 0)} \quad . \tag{2.14}$$

The factorized expansion in (2.13) is similar to (2.11), but with the difference that we have another sum running over all factors that depend on z_k. Consequently, in the resulting Implementation 4 with auxiliary neurons and dendritic branches there will be several groups of auxiliary neurons that connect to ν_k, where each group implements the expansion of one factor in (2.13). The alternative model that only uses dendritic computation (Implementation 5) will have groups of dendritic branches corresponding to the different factors. The number of auxiliary neurons that connect to ν_k in Implementation 4 (and the corresponding number of dendritic branches in Implementation 5) is equal to the sum of the exponents of the sizes of factors that depend on z_k: $\sum_{c \in C^k} 2^{D(\mathbf{z}^c_{\setminus k})}$, where $D(\mathbf{z}^c_{\setminus k})$ denotes the number of RVs in the vector $\mathbf{z}^c_{\setminus k}$. This number is never larger than $2^{|B_k|}$ (where $|B_k|$ is the size of the Markov blanket of z_k), which gives the corresponding number of auxiliary neurons or dendritic branches that are required in the Implementation 2 and 3. These two numbers can considerably differ in graphical models where the RVs participate in many factors, but the size of the factors is small. Therefore one advantage of this approach is that it requires in general fewer resources. On the other hand, it introduces a more complex connectivity between the auxiliary neurons and the principal neuron (compare Fig.2.5 with Fig.2.2). Furthermore, the network structure in Implementation 2 is compatible with a recently developed unsupervised learning architecture with spiking neurons that uses a local STDP learning rule (Nessler et al., 2010).

2.2.3.1 Implementation with Auxiliary Neurons and Dendritic Branches (Implementation 4)

A salient difference to the Markov blanket expansion and Implementation 2 arises from the fact that the r.h.s. of the factor expansion (2.13) contains an additional summation over all factors c that contain the RV z_k (we write C^k for this set of factors). This entails that the principal neuron ν_k has to sum up inputs from several groups of auxiliary neurons, one for each factor $c \in C_k$. Hence in contrast to Implementation 2, where the principal neuron fired whenever one of the associated auxiliary neurons fired, we now aim at satisfying the NCC by making sure that the membrane potential of ν_k approximates at any moment in time the r.h.s. of the NCC (2.4). One can achieve this by making sure that each auxiliary neuron $\alpha^k_{\mathbf{v}}$ fires immediately when the presynaptic principal neurons assume state $\mathbf{v}$. Some imprecision of the sampling may arise when the value of variables in $\mathbf{z}^c_{\setminus k}$ changes, while EPSPs caused by an earlier value of these variables have not yet vanished at the soma of ν_k. This problem can be solved if the firing of the auxiliary neuron caused by the new value of $\mathbf{z}^c_{\setminus k}$ shunts such EPSP, that had been caused by the preceding value of $\mathbf{z}^c_{\setminus k}$, directly in the corresponding dendrite. This shunting inhibition should have minimal effect on the membrane potential at the soma of ν_k. Therefore excitatory synaptic inputs from different auxiliary neurons $\alpha_{\mathbf{v}}$ (that cause a depolarization by an amount $w^{c,k}_{\mathbf{v}}$ at the soma) should arrive on different dendritic branches $\delta_{\mathbf{v}}$ of ν_k (see Fig. 2.5), that also have connections from associated inhibitory neurons $\iota_{\mathbf{v}}$.

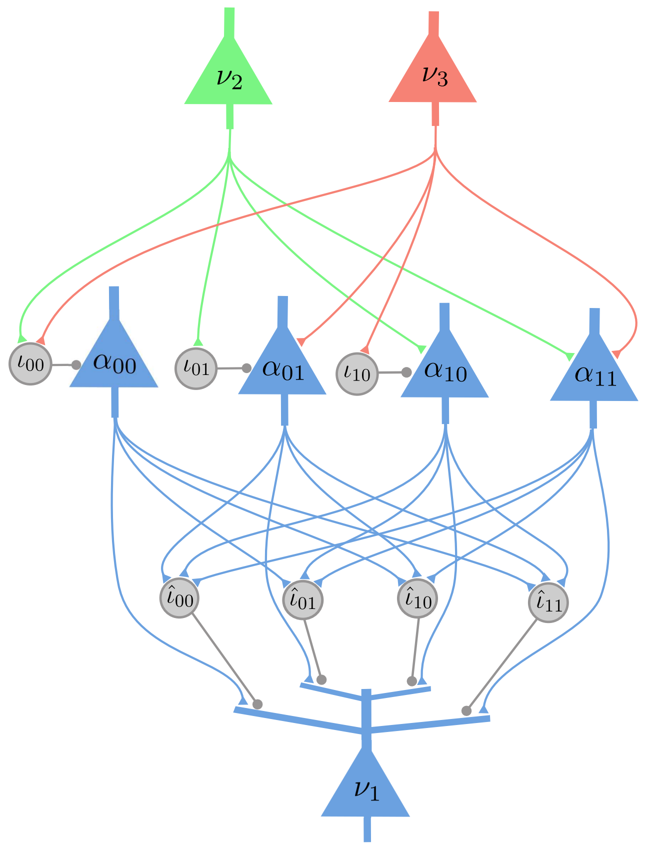

Figure 2.5: Implementation 4 (implementation with auxiliary neurons and dendritic branches, that uses the factorized expansion of the log-odd ratio) for the same explaining away motif as in Fig. 2.2 and 2.4. As in Fig. 2.2 there is one auxiliary neuron $\alpha_{\mathbf{v}}$ for each possible value assignment $\mathbf{v}$ to z_2 and z_3. The connections from the neurons ν_2 and ν_3 (that carry the current values of the RVs z_2 and z_3) to the auxiliary neurons are the same as in Fig. 2.2, and when these RVs change their value, the auxiliary neuron that corresponds to the new value fires. Each auxiliary neuron $\alpha_{\mathbf{v}}$ connects to the principal neuron ν_1 at a separate dendritic branch $\delta_{\mathbf{v}}$, and there is an inhibitory neuron $\hat{\iota}_{\mathbf{v}}$ connecting to the same branch. The rest of the auxiliary neurons connect to the inhibitory interneuron $\hat{\iota}_{\mathbf{v}}$. The function of the inhibitory neuron $\hat{\iota}_{\mathbf{v}}$ is to shunt the active EPSP caused by a recent spike from the auxiliary neuron $\alpha_{\mathbf{v}}$ when the value of the z_2 and z_3 changes from $\mathbf{v}$ to another value.

Fig. 2.5 shows the resulting implementation for the same explaining away motif of Fig. 2.1B as the precedings figures 2 and 3. Note that the RV z_1 occurs there only in a single factor $p(z_3|z_1, z_2)$, so that the previously mentioned summation of EPSPs from auxiliary neurons that arise from different factors cannot be demonstrated in this example.

2.2.3.2 Implementation with Dendritic Computation (Implementation 5)

The last neural implementation that we consider is an adaptation of Implementation 3 (the implementation with dendritic computation, that uses the Markov blanket expansion of the log-odd ratio) to the factorized expansion of the log-odd ratio. In this case each principal neuron, instead of having all its dendritic branches corresponding to different value assignments to the RVs of the Markov blanket, has several groups of dendritic branches, where each group corresponds to the linear expansion of one factor in the log-odd ratio in (2.13). Fig. 2.6 shows the complete spiking neural network that samples from the Bayesian network of Fig. 2.1B. The principal neuron ν_1 has the same structure and connectivity as in Implementation 3 (see Fig. 2.4), since the RV z_1 participates in only one factor, and the set of variables other that z_1 in this factor constitute the Markov blanket of z_1. The same is true for the principal neurons ν_3 and ν_4. As the RV z_2 occurs in two factors, the principal neuron ν_2 has two groups of dendritic branches, 4 for the factor $p(z_3|z_1, z_2)$ with synaptic input from the principal neurons ν_1 and ν_3, and 2 for the factor $p(z_4|z_2)$ with synaptic inputs from the principal neuron ν_4. Note for comparison, that this neuron ν_k needs to have 8 dendritic branches in Implementation 3, one for each assignment of values to the variables z_1, z_3 and z_4 in the Markov blanket of z_2.

The number of dendritic branches of a principal neuron ν_k in this implementation is the same as the number of auxiliary neurons for ν_k in Implementation 4, and is never larger than the number of dendritic branches of the neuron ν_k in Implementation 3. Although this implementation is more efficient with respect to the required number of dendritic branches, when considering the possible application of STDP for learning Implementation 3, the latter has the advantage that it can learn an approximate generative model of the probability distribution of the inputs without knowing apriori the factorization of the probability distribution.

The amplitude of the dendritic spikes from the dendritic branch $\delta_{\mathbf{v}}^{c,2}$ of the principal neuron ν_2 should be equal to the parameter $w_{\mathbf{v}}^{c,2}$ from (2.13). The index c identifies the two factors that depend on z_2. The membrane voltage at the soma of the principal neuron ν_2 is then equal to the sum of the contributions from the dendritic spikes of the active dendritic branches. At time t there is exactly one active branch in each of the two groups of dendritic branches. The sum of the contributions from the two active dendritic branches results in a membrane voltage at the soma of the principal neuron that corresponds to the r.h.s of the (2.13). In the Methods section we provide a general and detailed explanation of this approach.

2.2.4 Probabilistic Inference through Neural Sampling in Larger and More Complex Bayesian Networks

We have tested the viability of the previously described approach for neural sampling by satisfying the NCC also on two larger and more complex Bayesian networks: the well-known ASIA-network (Lauritzen and Spiegelhalter, 1988), and an even

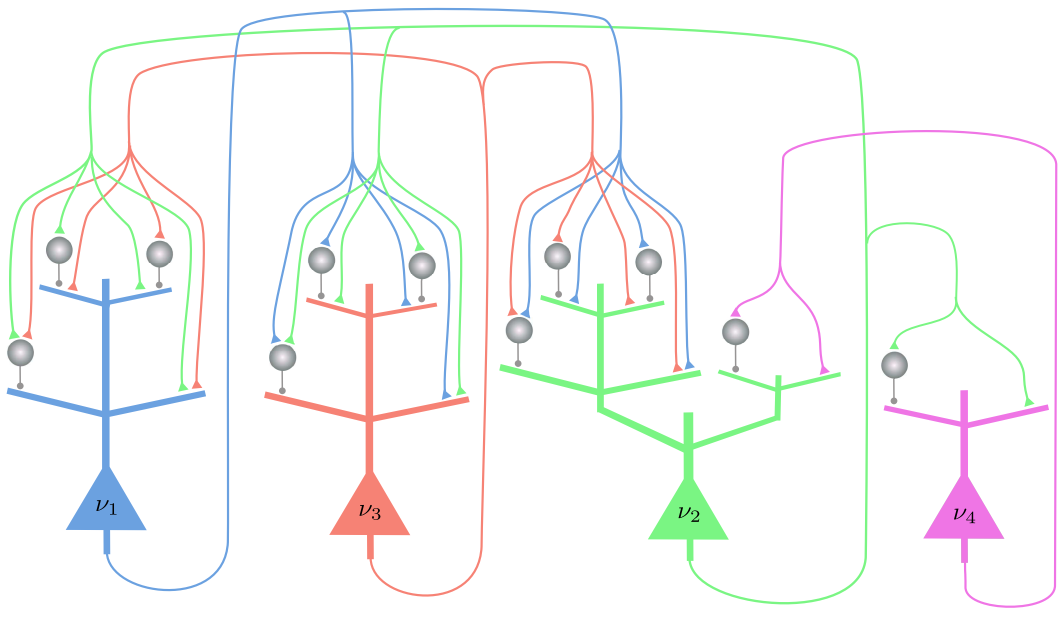

Figure 2.6: Implementation 5 (implementation with dendritic computation that is based on the factorized expansion of the log-odd ratio) for the Bayesian network shown in Fig. 2.1B. RV z_2 occurs in two factors, $p(z_3|z_1, z_2)$ and $p(z_4|z_2)$, and therefore ν_2 receives synaptic inputs from ν_1, ν_3 and ν_4 on separate groups of dendritic branches. Altogether the synaptic connections of this network of spiking neurons implement the graph structure of Fig. 2.1D.

larger randomly generated Bayesian network. The primary question is in both cases, whether the convergence speed of neural sampling is in a range where a reasonable approximation to probabilistic inference can be provided within the typical range of biological reaction times of a few 100 ms. In addition, we examine for the ASIA-network the question to what extent more complex and biologically more realistic shapes of EPSPs affect the performance. For the larger random Bayesian network we examine which difference in performance is caused by neuron models with absolute versus relative refractory periods.

2.2.4.1 Computer Simulation II: ASIA Bayesian Network

The ASIA-network is an example for a larger class of Bayesian networks that are of special interest from the perspective of Cognitive Science (Mansinghka et al., 2006). Networks of this type, that consist of 3 types of RVs (context information, true causes, observable symptoms) with directed edges only from one class to the next, capture the causal structure behind numerous domains of human reasoning. The ASIA-network (see Fig. 2.7A) encodes knowledge about direct influences between environmental factors, 3 specific diseases, and observable symptoms. A concrete distribution p that is compatible with this Bayesian network was specified through conditional probabilities for each node as in (Lauritzen and Spiegelhalter, 1988) (with one small change to avoid deterministic relationship among RVs, see Table 2

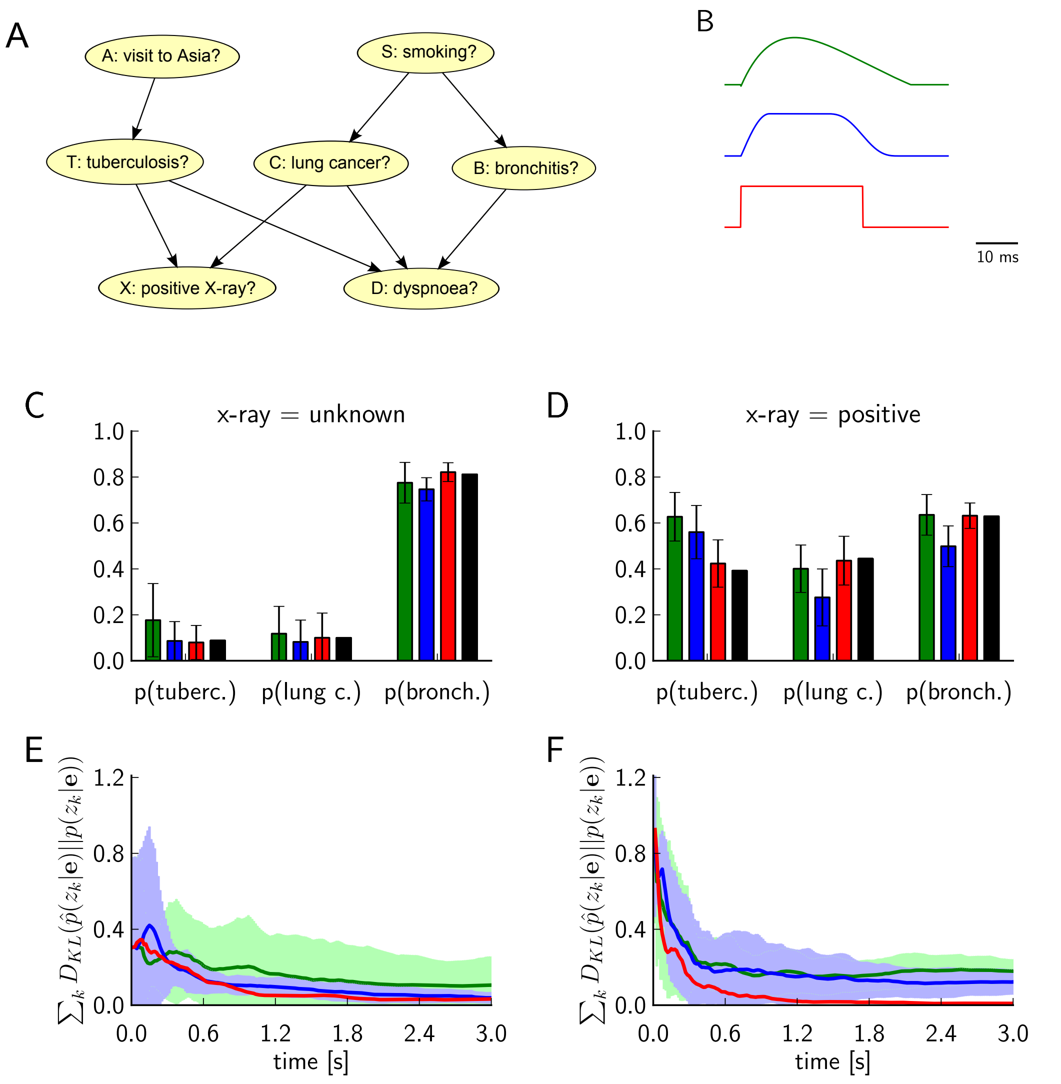

Figure 2.7: See next page for figure caption.

in Methods). The binary RVs of the network encode whether a person had a recent visit to Asia (A), whether the person smokes (S), the presence of diseases tuberculosis (T), lung cancer (C), and bronchitis (B), the presence of the symptom dyspnoea (D), and the result of a chest x-ray test (X). This network not only contains multiple “explaining away” effects (i.e., nodes with more than one parent), but also a loop (i.e., undirected cycle) between the RVs S, B, D, C. Hence no probabilistic inference approach based on belief propagation is guaranteed to work for this ASIA-network.

A typical example for probabilistic inference in this network arises when one enters as evidence the facts that the patient visited Asia (A = 1) and has Dyspnoea (D = 1), and asks what is the likelihood of each of the RVs T, C, B that represent the

Figure 2.7: Results of Computer Simulation II: Probabilistic inference in the ASIA network with networks of spiking neurons that use different shapes of EPSPs closer to neurophysiological measurements. The simulated neural networks correspond to Implementation 2. The evidence is changed at $t = 3$s from $\boldsymbol{e} = (A = 1, D = 1)$ to $\boldsymbol{e} = (A = 1, D = 1, X = 1)$ (by additionally clamping the x-ray test RV to 1). The probabilistic inference query is to estimate marginal posterior probabilities $p(T = 1|\boldsymbol{e})$, $p(C = 1|\boldsymbol{e}$, and $p(B = 1|\boldsymbol{e})$. **A)** The ASIA Bayesian network. **B)** The three different shapes of EPSPs used in the simulations, an alpha shape (green curve), a smooth plateau shape (blue curve) and the optimal rectangular shape (black curve). Panels **C)** and **D)** show the estimated marginal probabilities for each of the diseases, calculated from the samples generated during the first 800 ms of the simulation with alpha shaped (green bars), plateau shaped (blue bars) and rectangular (red bars) EPSPs, compared with the corresponding correct marginal posterior probabilities (black bars), for $\boldsymbol{e} = (A = 1, D = 1)$ in C) and $\boldsymbol{e} = (A = 1, D = 1, X = 1)$ in D). The results are averaged over 20 simulations with different random initial conditions and the error bars show the unbiased estimate of the standard deviation. Panels **E)** and **F)** show the sum of the Kullback-Leibler divergences between the correct and the estimated marginal posterior probability for each of the diseases using alpha shaped (green curve), plateau shaped (blue curve) and rectangular (red curve) EPSPs, for $\boldsymbol{e} = (A = 1, D = 1)$ in E) and $\boldsymbol{e} = (A = 1, D = 1, X = 1)$ in F). The results are averaged over 20 simulation trials, and the light green and light blue areas show the unbiased estimate of the standard deviation for the green and blue curves respectively (the standard deviation for the red curve is not shown to avoid clutter). The estimated marginal posteriors are calculated for each time point from the gathered samples from the beginning of the simulation (or from $t = 3$s for the second inference query with $\boldsymbol{e} = (A = 1, D = 1, X = 1)$).

diseases, and how the result of a positive x-ray test would affects these likelihoods.

We tested this probabilistic inference in a network of spiking neurons according to Implementation 2 with three different shapes of the EPSPs: an alpha EPSP, a plateau EPSP and the optimal rectangular EPSP (See Fig. 2.7A). These shapes match qualitatively the shapes of EPSPs recorded in the soma of pyramidal neurons for synaptic inputs that arrive on dendritic branches (see Fig. 2.1 in (Williams and Stuart, 2002)). The neurons in the spiking neural network had an absolute refractory period. Fig. 2.7C, D show that the network provides for all three shapes of the EPSPs within 800 ms of simulated biological time quite accurate answers to this probabilistic inference query. Fig. 2.7E, F show also with smoother shapes of the PSPs the networks arrive at good heuristic answers within several hundreds of milliseconds. The KL divergence converges in this case to a small non-zero value, indicating an error caused by the approximation.

Fig. 2.8 shows the spiking activity of the neural network with alpha shaped EPSPs in one of the simulation trials. During the first 3 seconds of the simulation the network alternated between two different modes of spiking activity, that correspond to two different modes of the posterior probability distribution. There are time periods when the principal neuron for the RV X (positive X-ray), T (tuberculosis) and C (lung c.) had a higher firing rate, with time periods in between where they were silent. After $t = 3$s, when the evidence that the x-ray test is positive was

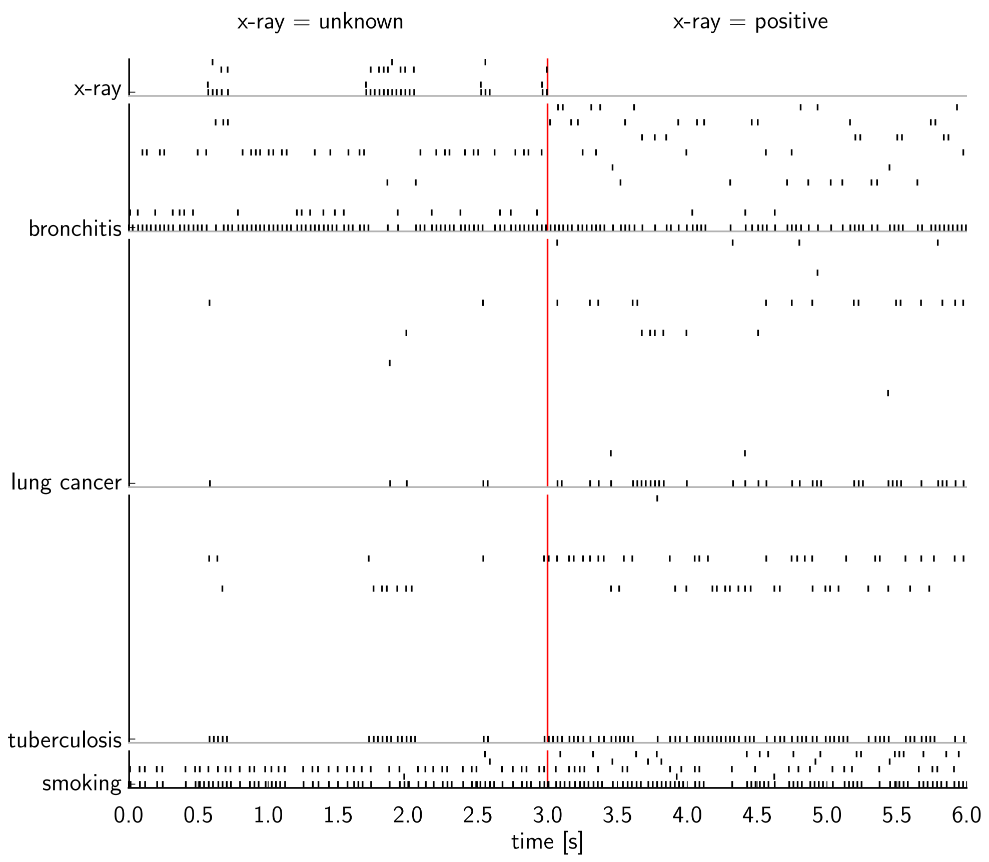

Figure 2.8: Spike raster of the spiking activity of the neurons in one of the 20 simulation trials described in Fig. 2.7 for the network of spiking neurons with alpha shaped EPSPs. The evidence was switched after 3 s (red vertical line) from $\boldsymbol{e} = (A = 1, D = 1)$ to $\boldsymbol{e} = (A = 1, D = 1, X = 1)$ (by clamping the RV X to 1). In each block of rows the lowest spike train shows the activity of a principal neuron (see left hand side for the label of the associated RV), and the spike trains above show the firing activity of the associated auxiliary neurons. After $t = 3$s the activity of the neurons for the x-ray test RV is not shown, since during this period the RV is clamped and the firing rate of its principal neuron is induced externally.

introduced, the activity of the network remained in the first mode.

2.2.4.2 Computer Simulation III: Randomly Generated Bayesian Network

In order to test the performance of neural sampling for an "arbitrary", less structured, and larger graphical model, we generated a random Bayesian network according to the method proposed in (Ide and Cozman, 2002) (the details of the generation algorithm are given in the Methods section). We added an additional constraint, that the maximum in-degree of the nodes should be not larger than 8. A resulting

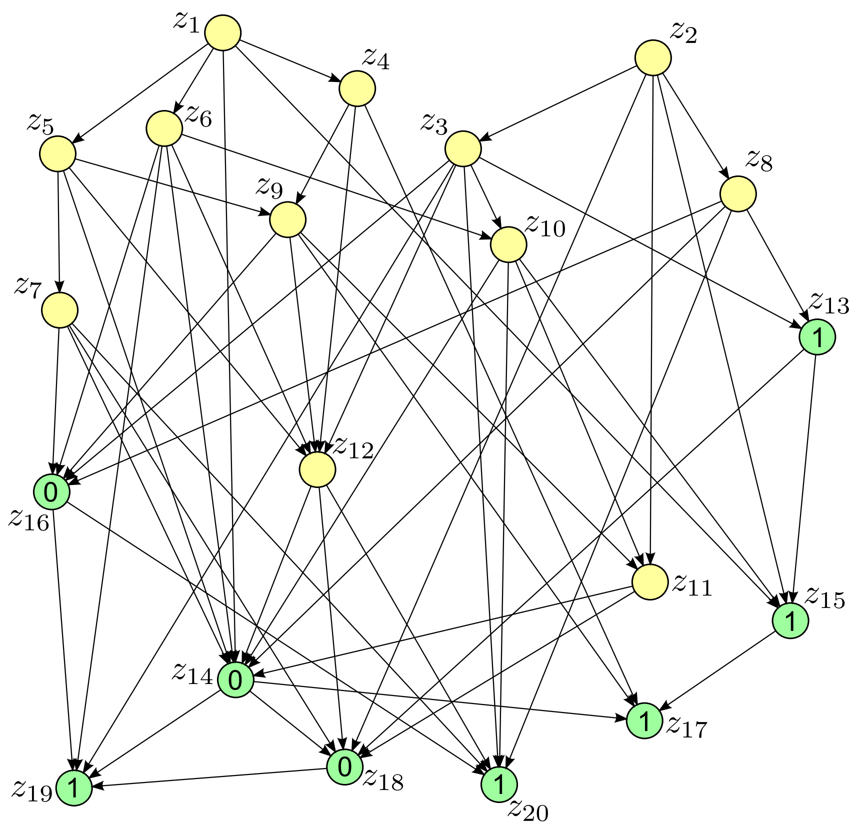

Figure 2.9: A randomly generated Bayesian network, for which a neural implementation of probabilistic inference was tested in Computer Simulation III. It contains 20 nodes. Each node has up to 8 parents. We consider the generic but more difficult instance for probabilistic inference where evidence $\boldsymbol{e}$ is entered for nodes $z_{13}, \ldots, z_{20}$ in the lower part of the directed graph. Conditional probability tables were also randomly generated for all RVs.

randomly generated network is shown in Fig. 2.9. It contains nodes with up to 8 parents, and it also contains numerous loops. For the RVs z_{13} to z_{20} we fixed a randomly chosen assignment $\boldsymbol{e}$. Neural sampling was tested for an ideal neural network that satisfies the NCC with a variety of random initial states, using spiking neurons with an absolute, and alternatively also with a relative refractory period.

Fig. 2.10A shows that in most of our 10 simulations with different randomly chosen initial states the sum of Kullback-Leibler divergences for the 12 RVs $z_1, \ldots, z_{12}$ becomes quite small within a second. Only in a few trials several seconds were needed for that. Fig. 2.10C and 2.10D show the spiking activity of the neural network from $t = 0$s to $t = 8$s in one of the 10 trials. It is interesting to observe that the network went through a number of network states, each of them characterized by a high firing rate of a particular subset of the neurons.

Similarly spontaneous switchings between internal network states have been reported in numerous biological experiments (see e.g. (Abeles et al., 1995; Miller and

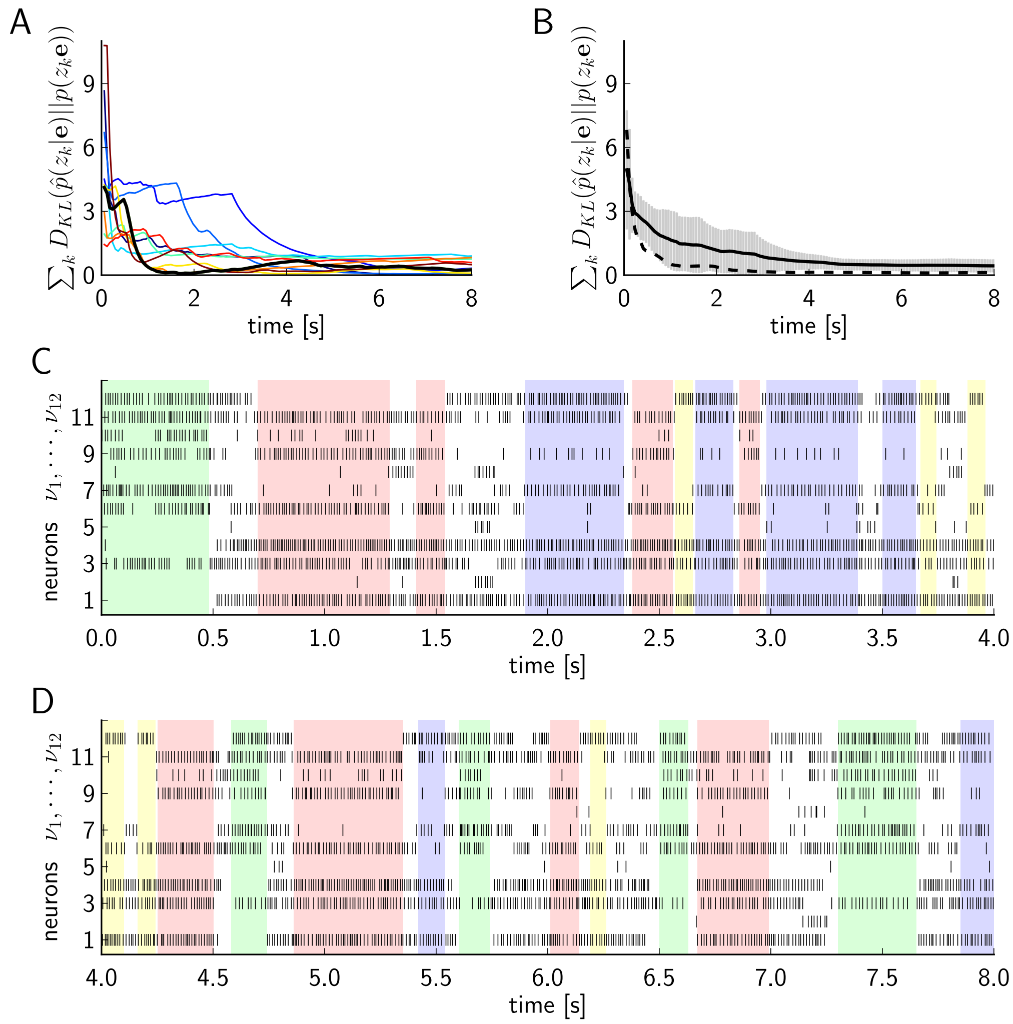

Figure 2.10: See next page for figure caption.

Katz, 2010)), but their functional role has remained unknown. In the context of Computer Simulation III these switchings between network states arise because this is the only way how this network of spiking neurons can sample from a multi-modal target distribution p.

2.3 Discussion

We have shown through rigorous theoretical arguments and computer simulations that networks of spiking neurons are in principle able to emulate probabilistic inference in general graphical models. The latter has emerged as a quite suitable mathematical framework for describing those computational tasks that artificial and

Figure 2.10: Computer Simulation III: Neural emulation of probabilistic inference through neural sampling in the fairly large and complex randomly chosen Bayesian network shown in Fig. 2.9. **A)** The sum of the Kullback-Leibler divergences between the correct and the estimated marginal posterior probability for each of the unobserved random variables $(z_1, z_2, \cdots, z_{12})$, calculated from the generated samples (spikes) from the beginning of the simulation up to the current time indicated on the x-axis, for simulations with a neuron model with relative refractory period. Separate curves with different colors are shown for each of the 10 trials with different initial conditions (randomly chosen). The bold black curve corresponds to the simulation for which the spiking activity is shown in C) and D). **B)** As in A) but the mean over the 10 trials is shown, for simulations with a neuron model with relative refractory period (solid curve) and absolute refractory period (dashed curve.). The gray area around the solid curve shows the unbiased estimate of the standard deviation calculated over the 10 trials. **C) and D)** The spiking activity of the 12 principal neurons during the simulation from $t = 0$ s to $t = 8$ s, for one of the 10 simulations (neurons with relative refractory period). The neural network enters and remains in different network states (indicated by different colors), corresponding to different modes of the posterior probability distribution.

biological intelligent agents need to solve. Hence the results of this work provide a link between this abstract description level of computational theory and models for networks of neurons in the brain. In particular, they provide a principled framework for investigating how nonlinear computational operations in network motifs of cortical microcircuits and in the dendritic trees of neurons contribute to brain computations on a larger scale.

We have presented three different theoretical approaches for extending the results of (Büsing et al., 2011), so that they yield explanations how probabilistic inference in general graphical models could be carried out through the inherent dynamics of recurrent networks of stochastically firing neurons (neural sampling). The first and simplest one was based on the fact that any distribution can be represented as marginal distribution of a 2^{nd} order Boltzmann distribution (2.5) with auxiliary RVs. However, as we have demonstrated in Fig.2.3, this approach yields rather slow convergence of the distribution of network states to the target distribution. This is a natural consequence of the deterministic definition of new RVs in terms of the original RVs, which reduces the conductance (Koller and Friedman, 2009; Levin et al., 2008) (i.e., the probability to get from one set of network states to another set of network states) of the Markov chain that is defined by the network dynamics. Further research is needed to clarify whether this deficiency can be overcome through other methods for introducing auxiliary RVs.

We have furthermore presented two approaches for satisfying the NCC (2.2) of (Büsing et al., 2011), which is a sufficient condition for sampling from a given distribution. These two closely related approaches rely on different ways of expanding the term on the r.h.s. of the NCC (2.4). The first approach can be used if the underlying graphical model implies that the Markov blankets of all RVs are relatively small. The second approach yields efficient neural emulations under a milder constraint: if each factor in a factorization of the target distribution is rather small (and if there are not too many factors). Each of these two approaches provides the theoretical basis for two different methods for satisfying the NCC in a network of spiking neurons: either through nonlinear computation in network motifs with auxiliary spiking neurons (that do not directly represent a RV of the target distribution), or through dendritic computation in multi-compartment neuron models. This yields altogether four different options for satisfying the NCC in a network of spiking neurons. These four options are demonstrated in Fig. 2.2 - 2.6 for a characteristic explaining away motif in the simple Bayesian network of Fig. 2.1B, that had previously been introduced to model inference in biological visual processing (Knill and Kersten, 1991). The second approach for satisfying the NCC never requires more auxiliary neurons or dendritic branches than the first approach.

Each of these four options for satisfying the NCC would be optimally supported by somewhat different features of the interaction of excitation and inhibition in canonical cortical microcircuit motifs, and by somewhat different features of dendritic computation. Sufficiently precise and general experimental data are not yet available for many of these features, and we hope that the computational consequences of these features that we have exhibited in this work will promote further experimental work on these open questions. In particular, the neural circuit of Fig. 2.5 uses an implementation strategy that requires for many graphical models (those where Markov blankets are substantially larger than individual factors) fewer auxiliary neurons. But it requires temporally precise local inhibition in dendritic branches that has negligible effects on the membrane potential at the soma, or in other dendritic branches that are used for this computation. Some experimental results in this direction are reported in (Williams and Stuart, 2003), where it was shown (see e.g. their Fig. 1) that IPSPs from apical dendrites of layer 5 pyramidal neurons are drastically attenuated at the soma. The options that rely on dendritic computation (Fig. 2.4 and 2.6) would be optimally supported if EPSPs from dendritic branches that are not amplified by dendritic spikes have hardly any effect on the membrane potential at the soma. Some experimental results which support this assumption for distal dendritic branches of layer 5 pyramidal neurons had been reported in (Williams and Stuart, 2002), see e.g. their Fig.1. With regard to details of dendritic spikes, these would optimally support the ideal theoretical models with dendritic computation if they would have a rather short duration at the soma, in order to avoid that they still affect the firing probability of the neuron when the state (i.e., firing or non-firing within the preceding time interval of length τ) of presynaptic neurons has changed. In addition, the ideal impact of a dendritic spike on the membrane potential at the soma would approximate a step function (rather than a function with a pronounced peak at the beginning).

We have focused in this work on the description of ideal neural emulations of probabilistic inference in general graphical models. Our results provide the basis for investigating how approximations to these ideal neural emulations could emerge through synaptic plasticity and other adaptive processes in neurons. First explorations of these questions suggest that in particular approximations to Implementations 1,2 and 4 could emerge through STDP in an ubiquitous network network motif of cortical microcircuits (Douglas and Martin, 2004b): Winner-Take-All circuits formed by populations of pyramidal neurons with lateral inhibition. This learning-based approach relies on the observation that STDP enables pyramidal neurons in the presence of lateral inhibition to specialize each on a particular pattern of presynaptic firing activity, and to fire after learning only when this presynaptic firing pattern appears (Nessler et al., 2010). These neurons would then assume the role of the auxiliary neurons, both in the first option with auxiliary RVs, and in the options shown in Fig. 2.2 and 2.5. Furthermore, the results of (Legenstein and Maass, 2011) suggest that STDP in combination with branch strength potentiation enables individual dendritic branches to specialize on particular patterns of presynaptic inputs, similarly as in the theoretically optimal constructions of Fig. 2.4 and 2.6. One difference between the theoretically optimal neural emulations and learning based approximations is that auxiliary neurons or dendritic branches learn to represent only the most frequently occurring patterns of presynaptic firing activity, rather than creating a complete catalogue of all theoretically possible presynaptic firing patterns. This has the advantage that fewer auxiliary neurons and dendritic branches are needed in these biologically more realistic learning-based approximations.

Other ongoing research explores neural emulations of probabilistic inference for non-binary RVs. In this case a stochastic principal neuron ν_k that represents a binary RV z_k is replaced by a Winner-Take-All circuit, that encodes the value of a multinomial or analog RV through population coding, see (Nessler et al., 2010).

2.3.1 Related Work

There are a number of studies proposing neural network architectures that implement probabilistic inference (Ackley et al., 1985; Hinton and Sejnowski, 1986; Deneve, 2008; Steimer et al., 2009; Litvak and Ullman, 2009; Rao, 2004, 2007; Bobrowski et al., 2009; Siegelmann and Holzman, 2010; Beck and Pouget, 2007; Rao and Ballard, 1999; Ma et al., 2008, 2006; Deneve et al., 2001; Yu and Dayan, 2005; Shi and Griffiths, 2009). Most of these models propose neural emulations of the belief propagation algorithm, where the activity of neurons or populations of neurons encodes intermediate values (called messages or beliefs) needed in the arithmetical calculation of the posterior probability distribution. With some exceptions (Deneve, 2008), most of the approaches assume rate-based coding of information and use rate-based neuron models or mean-field approximations.

In particular, in (Litvak and Ullman, 2009) a spiking neural network model was developed that performs the max-product message passing algorithm, a variant of belief propagation, where the necessary maximization and product operations were implemented by specialized neural circuits. A spiking neural implementation of the sum-product belief propagation algorithm was proposed in (Steimer et al., 2009), where the calculation and passing of the messages was achieved in a recurrent network of interconnected liquid state machines (Maass et al., 2002a). In these studies, that implemented probabilistic inference with spiking neurons through emulation of the belief propagation algorithm, the probability distributions or the messages during the calculation of the posterior distributions were encoded in an average firing rate of a population of neurons. Another interesting approach, that adopts an alternative spike-time based coding scheme, was described in (Deneve, 2008). In this study a spiking neuron model estimates the log-odd ratio of a hidden binary state in a hidden Markov chain, and it outputs a spike only when it receives new evidence from the inputs that causes a shift in the estimated log-odd ratio that exceeds a certain threshold, that is, only when new information about a change in the log-odd ratio is presented that cannot be predicted by the preceding spikes of the neuron. However, this study considers only a very restricted class of graphical models: Bayesian networks that are trees (where for example no explaining away can occur).

The idea that nonlinear dendritic mechanisms could account for the nonlinear processing that is required in neural models that perform probabilistic inference has been proposed previously in (Rao, 2007) and (Siegelmann and Holzman, 2010), albeit for the belief propagation algorithm. In (Rao, 2007) the authors introduce a neural model that implements probabilistic inference in hidden Markov models via the belief propagation algorithm, and suggest that the nonlinear functions that arise in the model can be mapped to the nonlinear dendritic filtering. In (Siegelmann and Holzman, 2010) another rate-based neural model that implements the loopy belief propagation algorithm in general graphical models was described, where the required multiplication operations in the algorithm were proposed to be implemented by the nonlinear processing in individual dendritic trees.

While there exist several different spiking neural network models in the literature that perform probabilistic inference based on the belief propagation algorithm, there is a lack of spiking neural network models that implement probabilistic inference through Markov chain Monte Carlo (MCMC sampling). To the best of our knowledge, the neural implementations proposed in this work are the only spiking neural networks for probabilistic inference via MCMC in general graphical models. In (Hinton and Sejnowski, 1986) a non-spiking neural network composed of stochastic binary neurons was introduced, that performs probabilistic inference via Gibbs sampling. The neural network in (Hinton and Sejnowski, 1986) performs inference via sampling in probability distributions that have only pairwise couplings between the RVs. An extension was proposed in (Sejnowski, 1987), that can perform Gibbs sampling in probability distributions with higher-order dependencies between the variables, which corresponds to the class of probability distributions that we consider in this work. A spiking neural network model based on the results in (Hinton and Sejnowski, 1986) had been proposed in (Hinton and Brown, 2000), for a restricted class of probability distributions that only have second order factors, and which satisfy some additional constraints on the conditional independencies between the variables. To the best of our knowledge, this approach had not been extended to more general probability distributions.

The existing gap between abstract computational models of brain processing that use MCMC algorithms for probabilistic inference on one hand, and neuroscientific data about neural structures and neural processes on the other hand, has been pointed out and emphasized by several studies (Hoyer and Hyvärinen, 2003; Gershman et al., 2009; Fiser et al., 2010). The results in (Büsing et al., 2011) and in this work propose neural circuit models that aim to bridge this existing gap, and thereby suggest new means for analysis and interpretations for both the computational models and experimental neuroscientific findings. For instance, perceptual multistability in ambiguous visual stimuli and several of its related phenomena were explained through abstract computational models that employ sequential sampling with the Metropolis MCMC algorithm (Gershman et al., 2009). In our simulations (see Fig. 2.10) we showed that a spiking neural network can exhibit multistability, where the state changes from one mode of the posterior distribution to another, even though the Markov chain defined by the neural network does not satisfy the detailed balance property like the Metropolis algorithm.

2.3.2 Experimentally Testable Predictions of our Models

Our models postulate that knowledge is encoded in the brain in the form of probability distributions p, that are not required to be of the restricted form of 2^{nd} order Boltzmann distributions (2.5). Furthermore they postulate that these distributions are encoded through synaptic weights and neuronal excitabilities, and possibly also through the strength of dendritic branches. Finally, our approach postulates that these learnt and stored probability distributions p are activated through the inherent stochastic dynamics of networks of spiking neurons, using nonlinear features of network motifs and neurons to represent higher order dependencies between RVs. It also predicts that (in contrast to the model of (Büsing et al., 2011)) synaptic connections between neurons are in general not symmetric, because this enables the network to encode higher order factors of p.

The postulate that knowledge is stored in the brain in the form of probability distributions, that are realized through the stochastic dynamics of neural circuits, is consistent with the ubiquitous trial-to-trial variability found in experimental data (Dean, 1981; Tolhurst et al., 1983). It has been partially confirmed through more detailed analyses, which show that spontaneous brain activity shows many characteristic features of brain responses to natural external stimuli ((Kenet et al., 2003; Raichle, 2010; Berkes et al., 2011)). Further analysis of spontaneous activity is needed in order to verify this prediction. Beyond this prediction regarding spontaneous activity, our approach proposes that fluctuating neuronal responses to external stimuli (or internal goals) represent samples from a conditional marginal distribution, that results from entering evidence $\boldsymbol{e}$ for a subset of RVs of the stored distribution p (see (2.1)). A verification of this prediction requires an analysis of the distributions of network responses – rather than just averaging – for repeated presentations of the same sensory stimulus or task. Similar analyses of human responses to repeated questions have already been carried out in cognitive science (Griffiths and Tenenbaum, 2006; Vul and Pashler, 2008; Denison et al., 2010), and have been interpreted as evidence that humans respond to queries by sampling from internally stored probability distributions.

Our resulting model for neural emulations of probabilistic inference predicts, that even strong firing of a single neuron (provided it represents a RV whose value has a strong impact on many other RVs) may drastically change the activity pattern of many other neurons (see the change of network activity after 3 s in Fig. 2.8, which results from a change in value of the RV that represents "x-ray"). One experimental result of this type had been reported in (Li et al., 2009). Fig. 2.8 also suggests that different neurons may have drastically different firing rates, where a few neurons fire a lot, and many others fire rarely. This is a consequence both of different marginal probabilities for different RVs, but also of the quite different computational role and dynamics of neurons that represent RVs ("principal neurons"), and auxiliary neurons that support the realization of the NCC, and which are only activated by a very specific activation patterns of other presynaptic neurons. Such strong differences in the firing activity of neurons has already been found in some experimental studies, see (Koulakov et al., 2009; Yassin et al., 2010). In addition, Fig. 2.10 predicts that recordings from multiple neurons can typically be partitioned into time intervals, where a different firing pattern dominates during each time interval (see (Abeles et al., 1995; Miller and Katz, 2010)) for some related experimental data.

Apart from these more detailed predictions, a central prediction of our model is, that a subset of cortical neurons (the "principal neurons") represent through their firing activity the current value of different salient RVs. This could be tested, for example, through simultaneous recordings from large numbers of neurons during experiments, where the values of several RVs that are relevant for the subject, and that could potentially be stored in the cortical area from which one records, are changed in a systematic manner.

It will be more difficult to test, which of the concrete implementations of computational preprocessing for satisfying the NCC that we have proposed, are implemented in some neural tissue. Both the underlying theoretical framework and our computer simulations (see Fig. 2.8) predict that the auxiliary neurons involved in these local computations are rarely active. More specifically, the model predicts that they only become active when some specific set of presynaptic neurons (whose firing state represents the current value of the RVs in $z_{\setminus k}$) assumes a specific pattern of firing and non-firing. Implementation 3 and 5 make corresponding predictions for the activity of different dendritic branches of pyramidal neurons, that could potentially be tested through Ca^{++}-imaging.

2.3.3 Conclusion

We have proposed a new modelling framework for brain computations, based on probabilistic inference. We have shown through computer simulations, that stochastic networks of spiking neurons can carry out demanding computational tasks within this modelling framework. This framework predicts specific functional roles for non-linear computations in network motifs and dendritic computation: They support learning and representation of higher order dependencies between salient random variables. On the micro level this framework proposes that local computational operations of neurons superficially resemble logical operations like AND and OR, but that these atomic computational operations are embedded into a stochastic network dynamics. Our framework proposes that the functional role of this stochastic network dynamics can be understood from the perspective of probabilistic inference through sampling from complex learnt probability distributions, that represent the knowledge base of the brain.

2.4 Methods

2.4.1 Markov Chains

A Markov chain $M = \langle S, T \rangle$ in discrete time is defined by a set S of states s (we consider for discrete time only the case where S has a finite size, denoted by $|S|$) together with a transition operator T. T is a conditional probability distribution $T(s|s')$ for the next state s of M, given its preceding state s'. The Markov chain M is started in some initial state $s(0)$, and moves through a trajectory of states $s(t)$ via iterated application of the stochastic transition operator T (more precisely, if $s(t-1)$ is the state at time $t-1$, then the next state $s(t)$ is drawn from the conditional probability distribution $T(s|s(t-1))$. A powerful theorem from probability theory (see e.g. p. 232 in (Grimmett and Stirzaker, 2001)) states that if M is irreducible (i.e., any state in S can be reached from any other state in S in finitely many steps with probability > 0) and aperiodic (i.e., its state transitions cannot be trapped in deterministic cycles), then the probability $p(s(t) = s|s(0)$ was the initial state) converges for $t \to \infty$ to a probability $p(s)$ that does not depend on $s(0)$. This state distribution p is called the stationary distribution of M. The irreducibility of M implies that p is the only distribution over the states S that is invariant under the transition operator T, i.e.

$$p(s) = \sum_{s' \in S} T(s|s') \cdot p(s') \quad . \tag{2.15}$$

Thus, in order to carry out probabilistic inference for a given distribution p, it suffices to construct an irreducible and aperiodic Markov chain M that leaves p invariant, i.e., satisfies (2.15). Analogous results hold for Markov chains in continuous time (Grimmett and Stirzaker, 2001)), on which we will focus in this work.

2.4.2 Neuron Models

We use two types of neurons, a stochastic point neuron model as in (Büsing et al., 2011), and a multi-compartment neuron model.

Point neuron model. We use the same point neuron model as in (Büsing et al., 2011). In (Büsing et al., 2011) rigorous proofs of the validity of neural sampling can only be given for spiking neurons with an absolute refractory period of length τ (the length of a PSP). The same holds for our results. But it was already shown in (Büsing et al., 2011) that practically also a variation of the neurons model with a relative refractory period can be used. In this variation of the model one can have a quite arbitrary refractory mechanism modeled with a refractory function $g(t)$, that represents the readiness of the neuron to fire within the refractory period. The firing probability of the neuron model is then

$$\rho(t) = f(u(t))g(t - \hat{t}) \qquad , \tag{2.16}$$

where $\hat{t}$ is the time of the last firing of the neuron before time t. The $g(t)$ function usually has value 0 for $g(0)$, meaning that the neuron cannot fire a second spike immediately after it has fired, and its value rises until $g(s) = 1$ for $s > \tau$, indicating that after time interval of duration τ the neuron fully recovers from its refractory period (this is a slight variation of the definition of g in (Büsing et al., 2011)).

For a given $g(t)$ function that models the refractory mechanism, the function $f(u)$ in the firing rate equation can be obtained as a solution from the equation

$$\forall u \in \mathbb{R} : f(u) \int_0^1 \exp\left(f(u) \int_0^r g(t)dt \right) dr = \exp(u) \qquad . \tag{2.17}$$

It can be shown that for any continuous function $g(t)$ there is a unique continuous function $f(u)$ that satisfies this equation (see (Büsing et al., 2011)). The multiplicative refractory function $g(t)$ together with a modified firing probability function $f(u)$ were derived in (Büsing et al., 2011) to ensure that each neuron performs correct local computations and generates correct samples from the desired probability distribution if one assumes that the other neurons do not change their state. This does not guarantee in the general case that the global computation of the network when all neurons operate simultaneously generates correct samples. However, as in (Büsing et al., 2011) we observed no significant deviations from the correct posteriors in our simulations.

Multi-compartment neuron model. For the neural implementations with dendritic computation (Implementations 3 and 5) we used a multi-compartment neuron model which is a modified version of the neuron model introduced in (Legenstein and Maass, 2011). It extends the stochastic point neuron model described above (with separate compartments that represent the dendritic branches) in order to capture the nonlinear effects in the integration of synaptic inputs at the dendritic branches of CA1 pyramidal neurons reported in (Losonczy et al., 2008) for radial oblique dendrites.

The local membrane voltage $A_i(t)$ of the branch i has a passive component $a_i(t)$ equal to the summation of the PSPs elicited by the spikes at the local synaptic inputs

$$a_i(t) = \sum_j w_{ij}\varepsilon_{ij}(t) \tag{2.18}$$

where w_{ij} is the synaptic efficacy of input j to branch i and $w_{ij}\varepsilon_{ij}(t)$ is the postsynaptic potential elicited in the branch i by the spikes from input j. We model $\varepsilon_{ij}(t)$ as

$$\varepsilon_{ij}(t) = \begin{cases} 1 & \text{if } t - \hat{t}_{ij} < \tau \\ 0 & \text{otherwise} \end{cases} \quad , \tag{2.19}$$

where $\hat{t}_{ij}$ is the time of the last spike before t that arrived at input j. If a synchronous synaptic input from many synapses at one branch exceeds a certain threshold, the membrane voltage at the branch exhibits a sudden jump due to regenerative integration processes resulting in a dendritic spike (Losonczy et al., 2008). This nonlinearity is modeled by a second active component $\hat{a}_i(t)$

$$\hat{a}_i(t) = \beta_i H(a_i(t) - \theta_i) \tag{2.20}$$

where $H(\cdot)$ denotes the Heaviside step function, and θ_i is the threshold of branch i. The branch potential $A_i(t)$ is equal to the sum of the passive component and the active component caused by the dendritic spike

$$A_i(t) = a_i(t) + \hat{a}_i(t) \quad . \tag{2.21}$$

The passive and active components contribute with a different weighting factor to the membrane potential at the soma. The passive component is conducted passively with a weighting factor $v_i < 1$ that models the attenuation of the passive signal. We assume in the neural implementations that the attenuation of the passive signal is strong, i.e. that $v_i \ll 1$. The dendritic spike is scaled by the branch strength $\hat{v}_i$. The membrane potential at the soma of the neuron is a sum of the active and passive contributions from all branches

$$u(t) = b + \sum_i v_i a_i(t) + \hat{v}_i \hat{a}_i(t) \tag{2.22}$$

The firing probability in this neuron model and its refractory mechanism are the same as for the point neuron model described above. It also can have an arbitrary refractory mechanism defined with the "readiness to fire" multiplicative function $g(t)$ and a modified firing probability $f(u)$.

2.4.3 Details to Second Order Boltzmann Distributions with Auxiliary Variables (Implementation 1)

Let $p(\mathbf{z})$ be a probability distribution

$$p(\mathbf{z}) = \frac{1}{Z} \prod_{f=1}^{F} \gamma_f(\mathbf{z}^f_{<3}) \prod_{c=1}^{C} \phi_c(\mathbf{z}^c) \tag{2.23}$$

that contains higher-order factors, where $\mathbf{z} = (z_1, z_2, \ldots, z_K)$ is a vector of binary RVs. $\gamma_f(\mathbf{z}^f)$ are the factors that depend on one or two RVs, and $\phi_c(\mathbf{z}^c)$ are the higher order factors that depend on more than 2 RVs. $\mathbf{z}^c$ is the vector of the RVs z_i in the factor $\phi_c(\mathbf{z}^c)$, $\mathbf{z}^f_{<3}$ is the vector of RVs z_i that the factor $\gamma_f(\mathbf{z}^f_{<3})$ depends on, and Z is the normalization constant. F is the number of first and second order factors, and C is the total number of factors of order 3 or higher. To simplify the notation, in the following we set $\gamma(\mathbf{z}) := \prod_{f=1}^{F} \gamma_f(\mathbf{z}^f_{<3})$, since this set of factors in $p(\mathbf{z})$ will not be changed in the extended probability distribution.

Auxiliary RVs are introduced for each of the higher order factors. Specifically, the higher-order relation of factor ϕ_c is represented by a set of auxiliary binary RVs $\mathbf{x}^c = \{x^c_{\mathbf{v}} | \mathbf{v} \in Z^c\}$, where we have a random variable $x^c_{\mathbf{v}}$ for each possible assignment $\mathbf{v} \in Z^c$ to the RVs in $\mathbf{z}^c$ (Z^c is the domain of values of the vector $\mathbf{z}^c$). With the additional sets of RVs $\mathbf{x}^c$ we define a probability distribution $p(\mathbf{z}, \mathbf{x})$ by

$$p(\mathbf{z}, \mathbf{x}) = \frac{1}{Z} \gamma(\mathbf{z}) \prod_c \left(\prod_{\mathbf{v} \in Z^c} \psi^c_{\mathbf{v}}(x^c_{\mathbf{v}}) \prod_{i \in \mathbf{I}^c} \beta^c_{\mathbf{v},i}(x^c_{\mathbf{v}}, z_i) \right) \quad . \tag{2.24}$$

We denote the ordered set of indices of the RVs that compose the vector $\mathbf{z}^c$ as $\mathbf{I}^c$, i.e.

$$\mathbf{I}^c = (i_1, i_2, \ldots, i_{|\mathbf{I}^c|}) \Leftrightarrow \mathbf{z}^c = (z_{i_1}, z_{i_2}, \ldots, z_{i_{|\mathbf{I}_c|}}) \quad , \tag{2.25}$$

where $|\mathbf{I}^c|$ denotes the number of indices in $\mathbf{I}^c$.

The second order factors $\beta^c_{\mathbf{v},i}(x, z)$ are defined as

$$\beta^c_{\mathbf{v},i}(x, z) = x \delta_{v_{(i)}, z} + (1 - x) \quad , \tag{2.26}$$

where $v_{(i)}$ denotes the component of the assignment $\mathbf{v}$ to $\mathbf{z}^c$ that corresponds to the variable z_i, and $\delta_{v_{(i)}, z_i}$ is the Kronecker-delta function. The factors $\beta^c_{\mathbf{v},i}(x^c_{\mathbf{v}}, z_i)$ represent a constraint that if the auxiliary RV $x^c_{\mathbf{v}}$ has value 1, then the values of the RVs in the corresponding factor $\mathbf{z}^c$ must be equal to the assignment $\mathbf{v}$ that $x^c_{\mathbf{v}}$ corresponds to. If all components of $\mathbf{x}^c$ are zero, then there is not any constraint on the $\mathbf{z}^c$ variables. This implies another property: at most one of the RVs $x^c_{\mathbf{v}}$ in the vector $\mathbf{x}^c$, the one that corresponds to the state of $\mathbf{z}^c$, can have value 1. Hence, the vector $\mathbf{x}^c$ can have two different states. Either all its RVs are zero, or exactly one component $x^c_{\mathbf{v}}$ is equal to 1, in which case one has $\mathbf{z}^c = \mathbf{v}$. The probability $p(\mathbf{z}, \mathbf{x})$ of states of $\mathbf{x}$ and $\mathbf{z}$ that do not satisfy these constraints is 0.

The values of the factors ϕ_c in $p(\mathbf{z})$ for various assignments of $\mathbf{z}^c$ are represented in $p(\mathbf{z}, \mathbf{x})$ by first-order factors that depend on a single one of the RVs $x^c_{\mathbf{v}}$. For each $x^c_{\mathbf{v}}$ we have a new factor with value $\psi^c_{\mathbf{v}}(x^c_{\mathbf{v}}) = \phi_c(\mathbf{v}) - 1$ if $x^c_{\mathbf{v}} = 1$, and $\psi^c_{\mathbf{v}}(x^c_{\mathbf{v}}) = 1$ otherwise. We assume that the original factors are first rescaled, so that $\phi_c(\mathbf{z}^c) > 1$ for all values of c and $\mathbf{z}^c$. We had to modify the values of the new factors by subtracting 1 from the original value $\phi_c(\mathbf{v})$, because we introduced an additional zero state for $\mathbf{x}^c$ that is consistent with any of the possible assignments of $\mathbf{z}^c$.

The resulting probability distribution $p(\mathbf{z}, \mathbf{x})$ consists of first and second order factors, and one can prove that it has the property

$$\sum_{\mathbf{x}} p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z}) \quad . \tag{2.27}$$

This can be seen as follows. If $p(\mathbf{z}, \mathbf{x}) \neq 0$, then for each c either $\mathbf{x}^c = \mathbf{0}$ (where $\mathbf{0}$ denotes the zero vector), or $\mathbf{x}^c$ has one component $x^c_{\mathbf{z}^c} = 1$, and $x^c_{\mathbf{v}} = 0$ for all $\mathbf{v} \neq \mathbf{z}^c$. The latter value of $\mathbf{x}^c$ is denoted by $\hat{\mathbf{x}}^c_{\mathbf{z}^c}$. For all other values of $\mathbf{x}^c$ we have $p(\mathbf{z}, \mathbf{x}) = 0$. Hence

$$\sum_{\mathbf{x}} p(\mathbf{z}, \mathbf{x}) = \sum_{\mathbf{x}^1 \in \{\mathbf{0}, \hat{\mathbf{x}}^1_{\mathbf{z}^c}\}} \sum_{\mathbf{x}^2 \in \{\mathbf{0}, \hat{\mathbf{x}}^2_{\mathbf{z}^c}\}} \cdots \sum_{\mathbf{x}^C \in \{\mathbf{0}, \hat{\mathbf{x}}^C_{\mathbf{z}^c}\}} p(\mathbf{z}, \mathbf{x}) \quad . \tag{2.28}$$

From the definition of the new factors ψ^c we have

$$p(\mathbf{z}, \mathbf{x}) = \frac{1}{Z} \gamma(\mathbf{z}) \prod_c \psi^c_{\mathbf{z}^c}(x^c_{\mathbf{z}^c}) = \frac{1}{Z} \gamma(\mathbf{z}) \prod_c (\phi_c(\mathbf{z}^c) - 1)^{x^c_{\mathbf{z}^c}} \quad . \tag{2.29}$$

Hence we can rewrite (2.28) as

$$\begin{aligned} \sum_{\mathbf{x}} p(\mathbf{z}, \mathbf{x}) &= \sum_{x^1_{\mathbf{z}^c} \in \{0,1\}} \sum_{x^2_{\mathbf{z}^c} \in \{0,1\}} \cdots \sum_{x^C_{\mathbf{z}^c} \in \{0,1\}} p(\mathbf{z}, \mathbf{x}) = \\ &= \sum_{x^1_{\mathbf{z}^c} \in \{0,1\}} \sum_{x^2_{\mathbf{z}^c} \in \{0,1\}} \cdots \sum_{x^C_{\mathbf{z}^c} \in \{0,1\}} \frac{1}{Z} \gamma(\mathbf{z}) \prod_c (\phi_c(\mathbf{z}^c) - 1)^{x^c_{\mathbf{z}^c}} = \\ &= \frac{1}{Z} \gamma(\mathbf{z}) \prod_c \phi_c(\mathbf{z}^c) = p(\mathbf{z}) \quad , \end{aligned} \tag{2.30}$$

yielding a proof of (2.27).

The resulting spiking neural network $\mathcal{N}$ consists of principal neurons ν_k, one for each of the original RVs z_k, and one principal neuron $\hat{\nu}^c_{\mathbf{v}}$ for each of the auxiliary RVs $x^c_{\mathbf{v}}$. If we assume that the factor ϕ_c depends on z_k, then the deterministic constraint that governs the relation between $\mathbf{z}$ and $\mathbf{x}$ is implemented by very strong excitatory connections M_{exc} (ideally equal to $+\infty$) between the principal neuron ν_k and all principal neurons $\hat{\nu}^c_{\mathbf{v}}$ for which z_k is 1 in the assignment $\mathbf{v}$ to $\mathbf{z}^c$. If for the principal neuron $\hat{\nu}^c_{\mathbf{v}}$ in the corresponding assignment $\mathbf{v}$ to $\mathbf{z}^c$ the value of z_k is 0, then there are strong inhibitory connections M_{inh} (ideally equal to $-\infty$) through an inhibitory interneuron between neuron ν_k and neuron $\hat{\nu}^c_{\mathbf{v}}$. Additionally, each of the principal neurons $\hat{\nu}^c_{\mathbf{v}}$ has a bias

$$b^c_{\mathbf{v}} = \log(\phi_c(\mathbf{v}) - 1) - \eta(\mathbf{v}) M_{exc} \quad , \tag{2.31}$$

where the function $\eta(\mathbf{v})$ denotes the number of coordinates of the vector $\mathbf{v}$ that have value 1. The biases of the principal neurons ν_k and the efficacies of the direct synaptic connections between the principal neurons ν_k that correspond to the second order factors in $p(\mathbf{z})$ are determined in the same way as for the spiking neural network structure in (Büsing et al., 2011) and depend only on the first and second order factors of $p(\mathbf{z})$.

We show in the following that the Markov chain represented by the spiking neural network that performs neural sampling in the 2^{nd} Boltzmann distribution $p(\mathbf{z}, \mathbf{x})$ is irreducible. We designate a state of the neural network with the vector $(\mathbf{z}, \boldsymbol{\zeta}, \mathbf{x}, \boldsymbol{\xi})$. Here $\boldsymbol{\zeta} = (\zeta_1, \zeta_2, \dots, \zeta_K)$, where ζ_k is the refractory variable of the principal neuron ν_k, and $\boldsymbol{\xi}$ is a vector of all refractory variables $\xi^c_{\mathbf{v}}$ for the principal neurons $\hat{\nu}^c_{\mathbf{v}}$ that correspond to the auxiliary RVs $x^c_{\mathbf{v}}$. The latter are defined as in (Büsing et al., 2011). At each spike of a corresponding neuron the refractory variable is set to τ (τ in neural sampling in discrete time is an integer number, that denotes the duration of the PSP in terms of discrete time steps). It decreases by 1 at each subsequent time step, until it reaches 0. We denote the transition operators for the refractory variables ζ_k changing from state $i+1$ to i with $T^k_{i,i+1}$, and changing from state 0 to τ with $T^k_{\tau,0}$. For the refractory variables $\xi^c_{\mathbf{v}}$ the transition operators are $T^{\mathbf{v},c}_{i,i+1}$ and $T^{\mathbf{v},c}_{\tau,0}$. In the proof we consider the ideal case where $M_{exc} \to +\infty$ and $M_{inh} \to -\infty$, which can result in infinitely large membrane potentials equal to $+\infty$ or $-\infty$. These values of the membrane potentials forbid the neuron to change the value of its RV, because if $u_k = +\infty$ then $T^k_{0,1} = 0$, and if $u_k = -\infty$ then $T^k_{\tau,0} = 0$ (see (Büsing et al., 2011) for details), and the neuron is locked to one value of the RV. In all other cases, when the value of the membrane potential remains finite, we have $T^k_{\tau,0} > 0$ and $T^k_{0,1} > 0$. In this case the principal neuron can reach any value of ζ_k from any other value in at most τ time steps. The same holds for the principal neurons $\hat{\nu}^c_{\mathbf{v}}$.

If we consider now an initial arbitrary non-forbidden state $(\bar{\mathbf{z}}, \bar{\boldsymbol{\zeta}}, \bar{\mathbf{x}}, \bar{\boldsymbol{\xi}})$, then each refractory variable $\xi^c_{\mathbf{v}}$ with $\mathbf{v} \neq \bar{\mathbf{z}}^c$ is equal to 0, and $\xi^c_{\mathbf{v}}$ with $\mathbf{v} = \bar{\mathbf{z}}^c$ can be either non-zero or 0. If $\xi^c_{\bar{\mathbf{z}}^c}$ is non-zero then, since the membrane potential of the principal neuron $\hat{\nu}^c_{\bar{\mathbf{z}}^c}$ is $\log(\phi_c(\bar{\mathbf{z}}^c) - 1)$, which is finite, there is a non-vanishing probability for the network state $(\bar{\mathbf{z}}, \bar{\boldsymbol{\zeta}}, \bar{\mathbf{x}}, \bar{\boldsymbol{\xi}})$ to change to another state in which $\xi^c_{\bar{\mathbf{z}}^c} = 0$ in at most τ time steps. Therefore we can conclude, that from the state $(\bar{\mathbf{z}}, \bar{\boldsymbol{\zeta}}, \bar{\mathbf{x}}, \bar{\boldsymbol{\xi}})$ we can reach the state $(\bar{\mathbf{z}}, \bar{\boldsymbol{\zeta}}, \mathbf{0}, \mathbf{0})$ that has $\mathbf{x} = \mathbf{0}$ and $\boldsymbol{\xi} = \mathbf{0}$ in at most τ time steps with a non-vanishing probability. In this new state all principal neurons ν_k are allowed to change the value of their RV, because their membrane potentials have finite values determined by the sum of their biases and the efficacies of the synaptic connections from the second order factors. Hence each non-zero ζ_k can change its value to 0 in at most τ time steps. From this it follows that from any non-forbidden state $(\bar{\mathbf{z}}, \bar{\boldsymbol{\zeta}}, \bar{\mathbf{x}}, \bar{\boldsymbol{\xi}})$ we can reach the zero state $(\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0})$ in at most 2τ time steps with non-vanishing probability. We proceed in a similar manner to prove that from the zero state we can reach any other non-forbidden state $(\tilde{\mathbf{z}}, \tilde{\boldsymbol{\zeta}}, \tilde{\mathbf{x}}, \tilde{\boldsymbol{\xi}})$. First we observe that from the zero state the principal neurons ν_k can change their states ζ_k to $\tilde{\zeta}_k$ in at most τ time steps, since they all have finite membrane potentials, i.e. we can reach the state $(\tilde{\mathbf{z}}, \tilde{\boldsymbol{\zeta}}, \mathbf{0}, \mathbf{0})$. Then in the state $(\tilde{\mathbf{z}}, \tilde{\boldsymbol{\zeta}}, \mathbf{0}, \mathbf{0})$ the principal neurons $\hat{\nu}^c_{\mathbf{v}}$ with $\mathbf{v} = \tilde{\mathbf{z}}^c$ have finite membrane potentials equal to $\log(\phi_c(\tilde{\mathbf{z}}^c) - 1)$, and they can change their states $\zeta^c_{\tilde{\mathbf{z}}^c}$ to $\tilde{\zeta}^c_{\tilde{\mathbf{z}}^c}$ in at most τ steps. Hence we have shown that we can reach any non-forbidden state $(\tilde{\boldsymbol{z}}, \tilde{\boldsymbol{\zeta}}, \tilde{\mathbf{x}}, \tilde{\boldsymbol{\xi}})$ from any other other non-forbidden state $(\bar{\mathbf{z}}, \bar{\boldsymbol{\zeta}}, \bar{\mathbf{x}}, \bar{\boldsymbol{\xi}})$ in at most 4τ steps with non-vanishing probability, i.e. the Markov chain is irreducible.

2.4.4 Details to Implementation 2

In this neural implementation each principal neuron ν_k has a dedicated preprocessing layer of auxiliary neurons with lateral inhibition. All neurons in the network are stochastic point neuron models.

The auxiliary neurons for the principal neuron ν_k receive as inputs the outputs of the principal neurons corresponding to all RVs in the Markov blanket of z_k. The number of auxiliary excitatory neurons that connect to the principal neuron ν_k is $2^{|B_k|}$ ($|B_k|$ is the number of elements of B_k), and we index these neurons with all possible assignments of values to the RVs in the vector $\mathbf{z}^{B_k}$. Thus, for each state $\mathbf{v}$ of values at the inputs $\mathbf{z}^{B_k}$ we have a corresponding auxiliary neuron $\alpha^k_{\mathbf{v}}$. The realization of the NCC is achieved by a specific connectivity between the inputs and the auxiliary neurons and appropriate values for the intrinsic excitabilities of the auxiliary neurons, so that at each moment in time only the auxiliary neuron $\alpha^k_{\mathbf{v}}$ corresponding to the current state of the inputs $\mathbf{z}^{B_k}(t) = \mathbf{v}$, if it is not inhibited by the lateral inhibition due to a recent spike from another auxiliary neuron, fires with a probability density as demanded by the NCC (2.3):

$$\rho_{\mathbf{v}}(t) = \frac{1}{\tau} \cdot \frac{p(z_k = 1|\mathbf{z}^{B_k} = \mathbf{v})}{p(z_k = 0|\mathbf{z}^{B_k} = \mathbf{v})}) \quad . \tag{2.32}$$

During the time when the state $\mathbf{v}$ of the inputs is active, the other auxiliary neurons are either strongly inhibited, or do not receive enough excitatory input to reach a significant firing probability.

The inputs connect to the auxiliary neuron $\alpha^k_{\mathbf{v}}$ either with a direct strong excitatory connection, or through an inhibitory interneuron $\iota^k_{\mathbf{v}}$ that connects to the auxiliary neuron. The inhibitory interneuron $\iota^k_{\mathbf{v}}$ fires whenever any of the principal neurons of the RVs $\mathbf{z}^{B_k}$ that connect to it fires. The auxiliary neuron $\alpha^k_{\mathbf{v}}$ receives synaptic connections according to the following rule: if the assignment $\mathbf{v}$ assigns a value of 1 to the RV z_i in the Markov blanket $\mathbf{z}^{B_k}$, then the principal neuron ν_i connects to the neuron with a strong excitatory synaptic efficacy $w^k_{\mathbf{v},i} = M^k_{\mathbf{v}}$, whereas if $\mathbf{v}$ assigns a value of 0 to z_i then the principal neuron ν_i connects to the inhibitory interneuron $\iota^k_{\mathbf{v}}$. Thus, whenever ν_i fires, the inhibitory interneuron fires and prevents the auxiliary neuron $\alpha^k_{\mathbf{v}}$ to fire for a time period τ. We will assume that the synaptic efficacy $M^k_{\mathbf{v}}$ is much larger than the log-odd ratio value of the RV z_k given $\mathbf{z}^{B_k} = \mathbf{v}$ according to the r.h.s. of (2.3). We set the bias of the auxiliary neuron $\alpha^k_{\mathbf{v}}$ equal to

$$b^k_{\mathbf{v}} = \log \frac{p(z_k = 1|\mathbf{z}^{B_k} = \mathbf{v})}{p(z_k = 0|\mathbf{z}^{B_k} = \mathbf{v})} - \eta(\mathbf{v}) M^k_{\mathbf{v}} \quad , \tag{2.33}$$

where $\eta(\mathbf{v})$ gives the number of components of the vector $\mathbf{v}$ that are 1.

If the value of the inputs at time t is $\mathbf{z}^{B_k}(t)$, and none of the neurons fired in the time interval $[t-\tau, t]$, then for an auxiliary neuron $\alpha^k_{\mathbf{v}}$ such that $\mathbf{v} \neq \mathbf{z}^{B_k}(t)$ there are two possibilities. Either there exists a component of $\mathbf{v}$ that is 0 and its corresponding input $z^{B_k}_i(t) = 1$, in which case the principal neuron of the RV $z^{B_k}_i$ connects to the inhibitory interneuron $\iota_{\mathbf{v}}^k$ and inhibits $\alpha_{\mathbf{v}}^k$. Or one has $\eta(\mathbf{z}^{B_k}(t)) < \eta(\mathbf{v})$ in which case the number of active inputs that connect to neuron $\alpha_{\mathbf{v}}^k$ do not provide enough excitatory input to reach the high threshold for firing. In this case the firing probability of the neuron $\alpha_{\mathbf{v}}^k$ is

$$\rho_{\mathbf{v}}^k(t) = \frac{1}{\tau}\exp\left(\log\frac{p(z_k = 1|\mathbf{z}^{B_k} = \mathbf{v})}{p(z_k = 0|\mathbf{z}^{B_k} = \mathbf{v})} - (\eta(\mathbf{v}) - \eta(\mathbf{z}^{B_k}(t))M_{\mathbf{v}}^k\right) \quad , \qquad (2.34)$$

and because of the strong synaptic efficacies of the excitatory connections equal to $M_{\mathbf{v}}^k$, which are by definition much larger than the log-odd ratio of the RV z_k, it is approximately equal to 0. Hence, only the neuron $\alpha_{\mathbf{v}}^k$ with $\mathbf{v} = \mathbf{z}^{B_k}(t)$ has a non-vanishing firing probability equal to (2.32).

The lateral inhibition between the auxiliary neurons is implemented through a common inhibitory circuit to which they all connect. The role of the lateral inhibition is to enforce the necessary refractory period of ν_k after any of the auxiliary neurons fires. When an auxiliary neuron fires, the inhibitory circuit is active during the duration of the excitatory PSP (equal to τ), and strongly inhibits the other neurons, preventing them from firing. The auxiliary neurons connect to the principal neuron ν_k with an excitatory connection strong enough to drive it to fire a spike whenever any one of them fires. During the time when the state of the input variables satisfies $\mathbf{z}^{B_k}(t) = \mathbf{v}$, the firing probability of the auxiliary neuron $\alpha_{\mathbf{v}}^k$ satisfies the NCC (2.3). This implies that the principal neuron ν_k satisfies the NCC as well.

Introducing an evidence of a known value of a RV in this model is achieved by driving the principal neuron with an external excitatory input to fire a spike train with a high firing rate when the observed value of the RV is 1, or by inhibiting the principal neuron with an external inhibitory input so that it remains silent when the observed value of the RV is 0.

2.4.5 Details to Implementation 3

We assume that the principal neuron ν_k has a separate dendritic branch $\delta_{\mathbf{v}}^k$ for each possible assignment of values to the RVs $\mathbf{z}^{B_k}$, and that the principal neurons corresponding to the RVs $\mathbf{z}^{B_k}$ in the Markov blanket B_k connect to these dendritic branches.

It is well known that synchronous activation of several synapses at one branch, if it exceeds a certain threshold, causes the membrane voltage at the branch to exhibit a sudden jump resulting from a dendritic spike. Furthermore the amplitude of such dendritic spike is subject to plasticity (Losonczy et al., 2008). We use a neuron model according to (Legenstein and Maass, 2011), that is based on these experimental data. The details of this multi-compartment neuron model were presented in the preceding subsection of Methods on Neuron Models. We assume in this model that the contribution of each dendritic branch to the soma membrane voltage is predominantly due to dendritic spikes, and that the passive conductance to the soma can be neglected. Thus, according to (2.22), the membrane potential at the soma is equal to the sum of the nonlinear active components contributed from each of the branches $\delta_{\mathbf{v}}^k$:

$$u_k(t) = b_k + \sum_{\mathbf{v}} \hat{v}_{\mathbf{v}}^k \hat{a}_{\mathbf{v}}^k(t) \qquad , \tag{2.35}$$

where $\hat{a}_{\mathbf{v}}^k(t)$ is the nonlinear contribution from branch $\delta_{\mathbf{v}}^k$, and $\hat{v}_{\mathbf{v}}^k$ is the strength of branch $\delta_{\mathbf{v}}^k$ (see (Losonczy et al., 2008) for experimental data on branch strengths). b_k is the target value of the membrane potential in the absence of any synaptic input. The nonlinear active component (dendritic spike) $\hat{a}_{\mathbf{v}}^k(t)$ is assumed to be equal to

$$\hat{a}_{\mathbf{v}}^k(t) = \beta_{\mathbf{v}}^k H(a_{\mathbf{v}}^k(t) - \theta_{\mathbf{v}}^k) \qquad , \tag{2.36}$$

where $H(\cdot)$ denotes the Heaviside step function, $a_{\mathbf{v}}^k(t)$ is the local activation, and $\theta_{\mathbf{v}}^k$ is the threshold of branch $\delta_{\mathbf{v}}^k$. The amplitude of the total contribution of branch $\delta_{\mathbf{v}}^k$ to the membrane potential at the soma is then $\hat{v}_{\mathbf{v}}^k \beta_{\mathbf{v}}^k$.

As can be seen in Fig. 2.4, the connectivity from the inputs to the dendritic branches is analogous as in Implementation 2 with auxiliary neurons: from each principal neuron ν_i so that z_i is in the Markov blanket of z_k there is a direct synaptic connection to the dendritic branch $\delta_{\mathbf{v}}^k$ if the assignment $\mathbf{v}$ assigns to z_i the value 1, or a connection to the inhibitory interneuron $\iota_{\mathbf{v}}^k$ in case $\mathbf{v}$ assigns the value 0 to z_i. The inhibitory interneuron $\iota_{\mathbf{v}}^k$ connects to its corresponding branch $\delta_{\mathbf{v}}^k$, and fires whenever any of the principal neurons that connect to it fire. The synaptic efficacies of the direct synaptic connections are assumed to satisfy the condition

$$\sum_{i \in S_{\mathbf{v}}^k} w_{\mathbf{v},i}^k > \theta_{\mathbf{v}}^k \qquad , \tag{2.37}$$

where $S_{\mathbf{v}}^k$ is the set of indices of principal neurons ν_i that directly connect to the dendritic branch $\delta_{\mathbf{v}}^k$, $w_{\mathbf{v},i}^k$ is the efficacy of the synaptic connection to the branch from ν_i, and $\theta_{\mathbf{v}}^k$ is the threshold at the dendritic branch for triggering a dendritic spike. Additionally, each synaptic weight $w_{\mathbf{v},i}^k$ should also satisfy the condition

$$w_{\mathbf{v},i}^k > \sum_{j \in S_{\mathbf{v}}^k} w_{\mathbf{v},j}^k - \theta_{\mathbf{v}}^k \qquad . \tag{2.38}$$

The same condition applies also for the efficacy $y_{\mathbf{v}}^k$ of the synaptic connection from inhibitory interneuron $\iota_{\mathbf{v}}^k$ to the dendritic branch $\delta_{\mathbf{v}}^k$.

These conditions ensure that if the current state of the inputs is $\mathbf{z}^{B_k}(t) = \mathbf{v}$, then the dendritic branch $\delta_{\mathbf{v}}^k$ will have an active dendritic spike, whereas all other dendritic branches do not receive enough total synaptic input to trigger a dendritic spike. The amplitude of the dendritic spike from branch $\delta_{\mathbf{v}}^k$ at the soma is

$$\hat{v}_{\mathbf{v}}^k \beta_{\mathbf{v}}^k = \log \frac{p(z_k = 1 | \mathbf{z}^{B_k} = \mathbf{v})}{p(z_k = 0 | \mathbf{z}^{B_k} = \mathbf{v})} + \lambda_k \qquad , \tag{2.39}$$

where λ_k is a positive constant that is larger than all possible negative values of the log-odd ratio. If the steady value of the membrane potential is equal to $b_k = -\lambda_k$, then we have at each moment a membrane potential that is equal to the sum of the amplitude of the nonlinear contribution of the single active dendritic branch and the steady value of the membrane potential, which yields the expression for the NCC (2.4).

2.4.6 Details to the Implementation 4

In this implementation a principal neuron ν_k has a separate group of auxiliary neurons for each factor c that depends on the variable z_k. The group of auxiliary neurons for the factor c receives inputs from the principal neurons that correspond to the set of the random variables $\mathbf{z}^c_{\setminus k}$ that factor c depends on, but without z_k. For each possible assignment of values $\mathbf{v}$ to the inputs $\mathbf{z}^c_{\setminus k}$, there is an auxiliary neuron in the group for the factor c, which we will denote with $\alpha^{c,k}_{\mathbf{v}}$. The neuron $\alpha^{c,k}_{\mathbf{v}}$ spikes immediately when the state of the inputs switches to $\mathbf{v}$ from another state, i.e. the spike marks the moment of the state change. This can achieved by setting the bias of the neuron similarly as in (2.33) to $b^k_{\mathbf{v}} = b_0 - \eta(\mathbf{v})M^k_{\mathbf{v}}$ where $\eta(\mathbf{v}))$ is the number of components of the vector $\mathbf{v}$ that are equal to 1, $M^k_{\mathbf{v}}$ is the efficacy of the direct synaptic connections from the principal neurons to $\alpha^{c,k}_{\mathbf{v}}$ and b_0 is a constant that ensures high firing probability of this neuron when the current value of the inputs is $\mathbf{v}$.

The connectivity from the auxiliary neurons to the principal neuron keeps the soma membrane voltage of the principal neuron ν_k equal to the log-odd ratio of z_k (= r.h.s. of (2.4)). From each auxiliary neuron $\alpha^{c,k}_{\mathbf{v}}$ there is one excitatory connection to the principal neuron, terminating at a separate dendritic branch $\delta^{c,k}_{\mathbf{v}}$. The efficacy of this synaptic connection is $\hat{w}^{c,k}_{\mathbf{v}} = w^{c,k}_{\mathbf{v}} + \lambda^c_k$, where $w^{c,k}_{\mathbf{v}}$ is the parameter from (2.13), and λ^c_k is a constant that shifts all these synaptic efficacies $\hat{w}^{c,k}_{\mathbf{v}}$ into the positive range.

Additionally, there is an inhibitory interneuron $\hat{\iota}^{c,k}_{\mathbf{v}}$ connecting to the same dendritic branch $\delta^{c,k}_{\mathbf{v}}$. The inhibitory interneuron $\hat{\iota}^{c,k}_{\mathbf{v}}$ receives input from all other auxiliary neurons in the same sub-circuit as the auxiliary neuron $\alpha^{c,k}_{\mathbf{v}}$, but not from $\alpha^{c,k}_{\mathbf{v}}$. The purpose of this inhibitory neuron is to shunt the active EPSP when the inputs $\mathbf{z}^c_{\setminus k}$ change their state from $\mathbf{v}$ to another state $\mathbf{v}'$. Namely, at the time moment when the inputs change to state $\mathbf{v}'$, the corresponding auxiliary neuron $\alpha^{c,k}_{\mathbf{v}'}$ will fire, and this will cause firing of the inhibitory interneuron $\hat{\iota}^{c,k}_{\mathbf{v}}$. A spike of the inhibitory interneuron should have just a local effect: to shunt the active EPSP caused by the previous state $\mathbf{v}$ at the dendritic branch $\delta^{c,k}_{\mathbf{v}}$. If there is not any active EPSP, this spike of the inhibitory interneuron should not affect the membrane potential at the soma of the principal neuron ν_k.

At any time t, each group of auxiliary neurons for a factor c contributes one EPSP to the principal neuron, through the synaptic input originating from the auxiliary neuron that corresponds to the current state of the inputs. The amplitude of the

EPSP from the sub-circuit that corresponds to the factor c is equal to $\hat{w}_{\mathbf{v}}^{c,k} = w_{\mathbf{v}}^{c,k} + \lambda_k^c$. If we assume that the bias of the soma membrane potential is $b_k = -\sum_{c \in C^k} \lambda_k^c$, then the total membrane potential at the soma of the principal neuron ν_k is equal to:

$$u_k(t) = b_k + \sum_{c \in C^k} (w_{\mathbf{v}}^{c,k} + \lambda_k^c) = \sum_{c \in C^k} w_{\mathbf{v}}^{c,k} \qquad , \tag{2.40}$$

which is equal to the expression on the r.h.s. of (2.13) when one assumes that $\mathbf{z}_{\setminus k}^c(t) = \mathbf{v}$. Hence, the principal neuron ν_k satisfies the NCC.

2.4.7 Details to the Implementation 5

In this implementation each principal neuron is a multi-compartment neuron of the same type as in Implementation 3, with a separate group of dendritic branches for each factor c in the probability distribution that depends on z_k. In the group c (corresponding to factor ϕ_c) there is a dendritic branch $\delta_{\mathbf{v}}^{c,k}$ for each assignment $\mathbf{v}$ to the variables $\mathbf{z}_{\setminus k}^c$ that the factor c depends on (without z_k). The dendritic branches in group c receive synaptic inputs from the principal neurons that correspond to the RVs $\mathbf{z}_{\setminus k}^c$. Each dendritic branch $\delta_{\mathbf{v}}^{c,k}$ can contribute a component $\hat{v}_{\mathbf{v}}^{c,k} \hat{a}_{\mathbf{v}}^{c,k}(t)$ to the soma membrane voltage $u_k(t)$ (where $\hat{v}_{\mathbf{v}}^{c,k}$ is like in Implementation 3 the branch strength of this branch), but only if the local activation $a_{\mathbf{v}}^{c,k}(t)$ in the branch exceeds the threshold for triggering a dendritic spike. The connectivity from the principal neurons corresponding to the RVs $\mathbf{z}_{\setminus k}^c$ to the dendritic branches of ν_k in the group c is such so that at time t only the dendritic branch corresponding to the current state of the inputs $\mathbf{z}_{\setminus k}^c(t)$ receives total synaptic input that crosses the local threshold for generating a dendritic spike and initiates a dendritic spike. This is realized with the same connectivity pattern from the inputs to the branches as in Implementation 3 depicted in Fig. 2.4. The amplitude of the dendritic spike of branch $\delta_{\mathbf{v}}^{c,k}$ at the soma should be $\hat{w}_{\mathbf{v}}^{c,k} = w_{\mathbf{v}}^{c,k} + \lambda_k^c$ where $w_{\mathbf{v}}^{c,k}$ is the parameter from (2.13) and λ_k^c is chosen as in Implementation 3.

The membrane voltage at the soma of the principal neuron ν_k is then equal to the sum of the dendritic spikes from the active dendritic branches. At time t there is exactly one active branch in each group of dendritic branches, the one which corresponds to the current state of the inputs. If we additionally assume that the bias of neuron ν_k is $b_k = -\sum_{c \in C^k} \lambda_k^c$, then the membrane voltage at the soma has the desired value (2.40).

2.4.8 Details to Computer Simulations

Details to Computer Simulation I. The simulations with the neural network that corresponds to the approach where the firing of the principal neurons satisfies the NCC were performed with the ideal version of the implementations 2, which assumes using rectangular PSPs and no delays in the synaptic connections. In the

Table 2.1: The conditional probability tables for the Bayesian network in Fig. 2.1B.

$p(z_3 = 1 \mid z_1, z_2)$	$z_1 = 0$	$z_1 = 1$
$z_2 = 0$	0.15	0.85
$z_2 = 1$	0.85	0.15

$p(z_4 = 1 \mid z_2)$	
$z_2 = 0$	0.15
$z_2 = 1$	0.85

simulation with the neural network that corresponds to Implementation 1, the network was also implemented with the ideal version of neural sampling. In both cases the duration of the rectangular PSPs was $\tau = 20$ms and the neurons had absolute refractory period of duration τ. The simulations lasted for 6 seconds biological time, where in the first 3 seconds the RV for the contour (z_4) was clamped to 1 and in the second 3 seconds clamped to 0. For each spiking neural network 10 simulation trials were performed, each time with different randomly chosen initial state. The values of the synaptic efficacies M_{exc} and M_{inh} in the simulation of implementation 1 were set to 10 times the largest value of any of the factors in the probability distribution. This ensures that a neuron with active input from a synapse with efficacy M_{exc} will have a very high membrane potential and will continuously stay active regardless of the state of the other inputs, and accordingly a neuron with active input from a synapse with efficacy M_{inh} will remain silent regardless of the state of the other inputs.

The values for the conditional probabilities $p(z_3|z_2, z_1)$ and $p(z_4|z_2)$ in the Bayesian network from Fig. 2.1 used in these simulations are given in Table 2.1. The prior probabilities $p(z_1 = 1)$ and $p(z_2 = 1)$ are both equal to 0.5.

Details to Computer Simulation II. The conditional probability tables of the ASIA-network are given in Table 2.2. We modified the original network from (Lauritzen and Spiegelhalter, 1988) by eliminating the "tuberculosis or cancer?" RV in order to get it in suitable form to be able to perform neural sampling in it. In the original ASIA network the "tuberculosis or cancer?" RV had deterministic links with the RVs "tuberculosis?" and "cancer?" which results in a Markov chain that is not connected. The model captures the following qualitative medical knowledge facts:

1. Shortness of breath or dyspnoea may be due to tuberculosis, lung cancer or bronchitis, none of them or many of them at the same time.
2. A recent visit to Asia increases the chance for tuberculosis.
3. Smoking is a risk factor for both lung cancer and bronchitis.

Table 2.2: The conditional probability tables for the ASIA Bayesian network.

$p(A=1)$	0.01
$p(S=1)$	0.5

$p(C=1\|S)$	
S = 0	0.01
S = 1	0.10

$p(T=1\|A)$	
A = 0	0.01
A = 1	0.05

$p(X=1\|T,C)$	C = 0	C = 1
T = 0	0.05	0.98
T = 1	0.98	0.98

$p(B=1\|S)$	
S = 0	0.3
S = 1	0.6

$p(D=1\|T,C,B)$	T = 0	T = 1
C = 0, B = 0	0.1	0.7
C = 0, B = 1	0.8	0.9
C = 1, B = 0	0.7	0.7
C = 1, B = 1	0.9	0.9

4. Tuberculosis and lung cancer significantly increase the chances of a positive chest x-ray test.

We used a point neuron model as in (Büsing et al., 2011) described in the Introduction section of this work, where the membrane potential of the neuron is a linear sum of the PSPs elicited by the input spikes. We performed all simulations with three different shapes for the EPSPs. The first EPSP was an alpha shaped EPSP curve $\varepsilon_1(t)$ defined as

$$\varepsilon_1(t) = \begin{cases} q_1 \cdot e(\frac{t}{\tau} + t_1) \cdot \exp(-(\frac{t}{\tau} + t_1)) - \frac{1}{2} & \text{if } 0 < t < (t_2 - t_1)\tau, \\ 0 & \text{otherwise.} \end{cases} , \quad (2.41)$$

where the t_1 and t_2 are the points in time where the alpha kernel $e \cdot t \cdot \exp(-t) = \frac{1}{2}$, $q_1 = 2.3$ is a scaling factor and $\tau = 17$ms is the time constant of the alpha kernel. The second used EPSP was a plateau shaped curve $\varepsilon_2(t)$ defined with the following equation

$$\varepsilon_2(t) = \begin{cases} q_2 \cdot (\sin(\frac{\pi t}{2\tau_s}) & \text{if } 0 < t < \tau_s, \\ q_2 & \text{if } \tau_s < t < \tau - \tau_e, \\ q_2 \cdot (\frac{\tau + \tau_e - t}{2\tau_e} - \frac{1}{2\pi}\sin(\frac{2\pi(\tau+\tau_e-t)}{2\tau_e})) & \text{if } \tau - \tau_e < t < \tau + \tau_e, \\ 0 & \text{otherwise.} \end{cases} , \quad (2.42)$$

where $\tau = 30$ms defines the duration of the EPSP and we use τ also to calculate the generated samples from the spike times. The $\tau_s = 7$ms defines the duration of the rise of the EPSP kernel after an input spike, $2\tau_e = 18$ms determines the duration of part of the EPSP curve corresponding to the fall of the PSP back to the baseline, modeled here with the sine function and $q_2 = 1.03$ is a scaling factor. The third shape of the EPSP that we used is the theoretically optimal rectangular shape with duration τ. All neurons have an absolute refractory period of duration τ.

The indirect connections going through inhibitory interneurons from the principal neurons to the auxiliary neurons are modeled as direct connections with negative synaptic efficacies with IPSPs that match the shape of the EPSPs described above. All synaptic connections in the network have delay equal to $d_{syn} = 0.1$ms. The excitatory synaptic weight from the principal neuron ν_i to an auxiliary neuron $\alpha^k_{\mathbf{v}}$ was set to

$$w^k_{\mathbf{v},i} = \max\left(\log\frac{p(z_k = 1|\mathbf{z}^{B_k} = \mathbf{v})}{p(z_k = 0|\mathbf{z}^{B_k} = \mathbf{v})} + 10, 0\right) \quad , \tag{2.43}$$

and the synaptic weight for the inhibitory synaptic connection from the principal neuron ν_i to an auxiliary neuron $\alpha^k_{\mathbf{v}}$ (which models the indirect inhibitory connection through the inhibitory interneuron $\iota^k_{\mathbf{v}}$) is set to

$$w^k_{\mathbf{v},i} = \min\left(-10 - \log\frac{p(z_k = 1|\mathbf{z}^{B_k} = \mathbf{v})}{p(z_k = 0|\mathbf{z}^{B_k} = \mathbf{v})}, 0\right) \quad . \tag{2.44}$$

The efficacy of the synaptic connections from the auxiliary neurons to their principal neuron are set to $w_{ap} = 30$. The lateral inhibition is implemented by a single inhibitory neuron that receives excitatory connections from all auxiliary neurons with synaptic efficacy equal to $w_{ai} = 30$. The inhibitory neuron connects back to all auxiliary neurons and these synaptic connections have rectangular shaped IPSPs with duration $\tau_i = 30$ ms. These rectangular IPSPs approximate the effect that a circuit of fast-spiking bursting inhibitory neuron with short IPSPs would have on the membrane potential of the auxiliary neurons. The efficacy of the synaptic connection from the inhibitory neuron for the lateral inhibition to the auxiliary neuron $\alpha^k_{\mathbf{v}}$ is set equal to $w^k_{\mathbf{v},i}$ in the previous equation. The bias of the principal neurons are set to $b = -10$ and the biases of the auxiliary neurons are set according to (2.33). The inhibitory interneuron for the lateral inhibition has bias $b = -10$.

The evidence about known random variables in the neural network was introduced by injected constant current in the corresponding principal neurons of amplitude $A_+ = 40$ if the value of the RV is 1 and $A_- = -40$ if the value of the RV is 0. The simulations were performed for $T_{sim} = 6$ sec. biological time. For the separate cases of each EPSP shape the results were averaged over 20 simulation trials with different initial states of the spiking neural network and different noise through the simulation. The initial states were randomly chosen from the prior distribution of the ASIA network which corresponds to a random state in the activity of the spiking network when no evidence is introduced. For control we performed the same simulations with randomly chosen initial states from an uniform distribution, and the results showed slightly slower convergence (data not shown). The initial states were set by injecting constant current pulse in the principal neurons at the beginning of the simulation, for the unknown RVs with amplitude $A_+ = 40$ ($A_- = -40$) if the value of the RV in the initial state is 1 (0) and duration equal to $\tau_{init} = 15$ms.

The simulations in Computer Simulation II were performed with the PCSIM[2] simulator for neural circuits (Pecevski et al., 2009).

Details to Computer Simulation III. The simulations were performed with the ideal implementation of the NCC, which corresponds to using rectangular PSPs and zero delays in the synaptic connections in the implementations 2-5. We performed 10 simulations with an implementation that uses the neuron model relative refractory period and another 10 simulations with an implementation that uses the neuron model with absolute refractory period. The duration of the PSPs was $\tau = 20$ ms.

The Bayesian network in this simulation was randomly generated with a variation of the Markov chain Monte Carlo sampling algorithm proposed in (Ide and Cozman, 2002). Instead of allowing arcs in the Bayesian network in both directions between the nodes and checking at each new iteration whether the generated Bayesian network graph is acyclic like in (Ide and Cozman, 2002), we preserved an ordering of the nodes in the graph and allow an edge from the node z_i to the node z_j only if $i < j$. We started with a simple connected graph where each node z_i, except for the first node z_1, has connection from node z_{i-1}. We then performed the following MCMC iterations

1. Choose randomly a pair of nodes (z_i, z_j) where $i < j$;
2. If there is an edge from z_i to z_j then remove the edge if the Bayesian network remains connected, otherwise keep the same Bayesian network from the previous iteration;
3. If there is not an arc, then create an edge from z_i to z_j if the node z_j has less than 8 parents, otherwise keep the Bayesian network from the previous iteration.

Similarly to the proofs in (Ide and Cozman, 2002), one can prove that the stationary distribution of the above Markov chain is a uniform distribution over all valid Bayesian networks that satisfy the constraint that a node can not have more than 8 parents. To generate the Bayesian network used in the simulations we performed 500000 iterations of the above Markov chain. The conditional probability distributions for the Bayesian network were sampled from Dirichlet distributions with priors $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ with $\alpha_i = 0.6$ for all i.

In the simulations that use a neuron model with a relative refractory mechanism, we used the following form for the refractory function $g_k(t)$

$$g(t) = \frac{t}{\tau} - \frac{\sin(\frac{2\pi t}{\tau})}{2\pi} \quad . \tag{2.45}$$

The corresponding function $f(u)$ for the firing probability is defined implicitly by (2.17).

[2]web site: www.igi.tugraz.at/pcsim

2.5 Acknowledgements

This chapter is based on the paper *Probabilistic Inference in General Graphical Models through Sampling in Stochastic Networks of Spiking Neurons* by Dejan Pecevski (DP), Lars Büsing (LB) and Wolfgang Maass (WM). The paper was submitted for publication in 2011 and is under review. The experiments in the paper were concieved and designed by DP and WM. DP conducted the experiments and analysed the simulation results. The paper builds on the theory of neural sampling developed by LB and reported in (Büsing et al., 2011). DP and WM provided the additional theoretical derivations and analysis in the paper. DP and WM wrote the manuscript. LB provided valuable comments that helped to improve the manuscript.

Chapter 3

A Learning Theory for Reward-Modulated Spike-Time-Dependent Plasticity with Application to Biofeedback

Contents

Reward-modulated spike-timing-dependent plasticity (STDP) has recently emerged as a candidate for a learning rule that could explain how behaviorally relevant adaptive changes in complex networks of spiking neurons could be achieved in a self-organizing manner through local synaptic plasticity. However the capabilities and limitations of this learning rule could so far only be tested through computer simulations. This work provides tools for an analytic treatment of reward-modulated STDP, which allows us to predict under which conditions reward-modulated STDP will achieve a desired learning effect. These analytical results imply that neurons can learn through reward-modulated STDP to classify not only spatial, but also temporal firing patterns of presynaptic neurons. They also can learn to respond to specific presynaptic firing patterns with particular spike patterns. Finally, the resulting learning theory predicts that even difficult credit-assignment problems, where it is very hard to tell which synaptic weights should be modified in order to increase the global reward for the system, can be solved in a self-organizing manner through reward-modulated STDP. This yields an explanation for a fundamental experimental result on biofeedback in monkeys by Fetz and Baker. In this experiment monkeys were rewarded for increasing the firing rate of a particular neuron in the cortex, and were able to solve this extremely difficult credit assignment problem. Our model for this experiment relies on a combination of reward-modulated STDP with variable spontaneous firing activity. Hence it also provides a possible functional

explanation for trial-to-trial variability, which is characteristic for cortical networks of neurons, but has no analogue in currently existing artificial computing systems. In addition our model demonstrates that reward-modulated STDP can be applied to all synapses in a large recurrent neural network without endangering the stability of the network dynamics.

3.1 Introduction

Numerous experimental studies (see Abbott and Nelson (2000) for a review; Jacob et al. (2007) discusses more recent in-vivo results) have shown that the efficacy of synapses changes in dependence of the time difference $\Delta t = t_{post} - t_{pre}$ between the firing times t_{pre} and t_{post} of the pre- and postsynaptic neurons. This effect is called spike-timing-dependent plasticity (STDP). But a major puzzle for understanding learning in biological organisms is the relationship between experimentally well-established rules for STDP on the microscopic level, and adaptive changes of the behavior of biological organisms on the macroscopic level. Neuromodulatory systems, which send diffuse signals related to reinforcements (rewards) and behavioral state to several large networks of neurons in the brain, have been identified as likely intermediaries that relate these two levels of plasticity. It is well-known that the consolidation of changes of synaptic weights in response to pre- and postsynaptic neuronal activity requires the presence of such third signals Bailey et al. (2000); Gu (2002). In particular, it has been demonstrated that dopamine (which is behaviorally related to novelty and reward prediction Schultz (2007)) gates plasticity at corticostriatal synapses Reynolds et al. (2001); Reynolds and Wickens (2002) and within the cortex Bao et al. (2001). It has also been shown that acetylcholine gates synaptic plasticity in the cortex (see for example Shulz et al. (2000) and Thiel et al. (2002); Shulz et al. (2003) contains a nice review of the literature).

Corresponding spike-based rules for synaptic plasticity of the form

$$\frac{d}{dt}w_{ji}(t) = c_{ji}(t)d(t) \tag{3.1}$$

have been proposed in Izhikevich (2007) and Florian (2007) (see Fig. 3.1 for an illustration of this learning rule), where w_{ji} is the weight of a synapse from neuron i to neuron j, $c_{ji}(t)$ is an eligibility trace of this synapse which collects weight changes proposed by STDP, and $d(t) = h(t) - \bar{h}$ results from a neuromodulatory signal $h(t)$ with mean value $\bar{h}$. It was shown in Izhikevich (2007) that a number of interesting learning tasks in large networks of neurons can be accomplished with this simple rule (3.1). It has recently been shown that quite similar learning rules for spiking neurons arise when one applies the general framework of distributed reinforcement learning from Baxter and Bartlett (1999) to networks of spiking neurons Baras and Meir (2007); Florian (2007), or if one maximizes the likelihood of postsynaptic firing at desired firing times Pfister et al. (2006). However no analytical tools have been available, which make it possible to predict for what learning tasks, and under which parameter settings, reward-modulated STDP will be successful. This work provides such analytical tools, and demonstrates their applicability and significance through a variety of computer simulations. In particular, we identify conditions under which neurons can learn through reward-modulated STDP to classify temporal presynaptic firing patterns, and to respond with particular spike patterns.

We also provide a model for the remarkable operant conditioning experiments of Fetz and Baker (1973) (see also Fetz (1969, 2007)). In the simpler ones of these experiments the spiking activity of single neurons (in area 4 of the precentral gyrus of monkey cortex) was recorded, the deviation of the current firing rate of an arbitrarily selected neuron from its average firing rate was made visible to the monkey through the displacement of an illuminated meter arm, whose rightward position corresponded to the threshold for the feeder discharge. The monkey received food rewards for increasing (or in alternating trials for decreasing) the firing rate of this neuron. The monkeys learnt quite reliably (within a few minutes) to change the firing rate of this neuron in the currently rewarded direction.[1] Obviously the existence of learning mechanisms in the brain which are able to solve this extremely difficult credit assignment problem provides an important clue for understanding the organization of learning in the brain. We examine in this work analytically under what conditions reward-modulated STDP is able to solve such learning problem. We test the correctness of analytically derived predictions through computer simulations of biologically quite realistic recurrently connected networks of neurons, where an increase of the firing rate of one arbitrarily selected neuron within a network of 4000 neurons is reinforced through rewards (which are sent to all 142813 synapses between excitatory neurons in this recurrent network). We also provide a model for the more complex operant conditioning experiments of Fetz and Baker (1973) by showing that pairs of neurons can be differentially trained through reward-modulated STDP, where one neuron is rewarded for increasing its firing rate, and simultaneously another neuron is rewarded for decreasing its firing rate. More precisely, we increased the reward signal $d(t)$ which is transmitted to all synapses between

[1]Adjacent neurons tended to change their firing rate in the same direction, but also differential changes of directions of firing rates of pairs of neurons are reported in Fetz and Baker (1973) (when these differential changes were rewarded). For example, it was shown in Fig. 3.9 of Fetz and Baker (1973) (see also Fig. 3.1 in Fetz (2007)) that pairs of neurons that were separated by no more than a few hundred microns could be independently trained to increase or decrease their firing rates. It was also reported in Fetz and Baker (1973), and further examined in Fetz and Finocchio (1975), that bursts of the reinforced neurons were often accompanied by activations of specific muscles. But the relationship between bursts of the recorded neurons in precentral motor cortex and muscle activations was reported to be quite complex and often dropped out after continued reinforcement of the neuron alone. Furthermore in Fetz and Finocchio (1975) it was shown that all neurons tested in that study could be dissociated from their correlated muscle activity by differentially reinforcing simultaneous suppression of EMG activity. These results suggest that the solution of the credit assignment problem by the monkeys (to stronger activate that neuron out of billions of neurons in their precentral gyrus that was reinforced) may have been supported by large scale exploration strategies that were associated with muscle activations. But the previously mentioned results on differential reinforcements of two nearby neurons suggest that this large scale exploration strategy had to be complemented by exploration on a finer spatial scale that is difficult to explain on the basis of muscle activations (see section 3.2 of Fetz (2007) for a detailed discussion).

excitatory neurons in the network whenever the first neuron fired, and decreased this reward signal whenever the second neuron fired (the resulting composed reward corresponds to the displacement of the meter arm that was shown to the monkey in these more complex operant conditioning experiments).

Our theory and computer simulations also show that reward-modulated STDP can be applied to all synapses within a large network of neurons for long time periods, without endangering the stability of the network. In particular this synaptic plasticity rule keeps the network within the asynchronous irregular firing regime, which had been described in Brunel (2000) as a dynamic regime that resembles spontaneous activity in the cortex. Another interesting aspect of learning with reward-modulated STDP is that it requires spontaneous firing and trial-to-trial variability within the networks of neurons where learning takes place. Hence our learning theory for this synaptic plasticity rule provides a foundation for a functional explanation of these characteristic features of cortical network of neurons that are undesirable from the perspective of most computational theories.

3.2 Results

We first give a precise definition of the learning rule (3.1) for reward-modulated STDP. The standard rule for STDP, which specifies the change $W(\Delta t)$ of the synaptic weight of an excitatory synapse in dependence on the time difference $\Delta t = t_{post} - t_{pre}$ between the firing times t_{pre} and t_{post} of the pre- and postsynaptic neuron, is based on numerous experimental data (see Abbott and Nelson (2000)). It is commonly modeled by a so-called learning curve of the form

$$W(\Delta t) = \begin{cases} A_+ e^{-\Delta t/\tau_+} & , \quad \text{if } \Delta t \geq 0 \\ -A_- e^{\Delta t/\tau_-} & , \quad \text{if } \Delta t < 0 \end{cases} , \tag{3.2}$$

where the positive constants A_+ and A_- scale the strength of potentiation and depression respectively, and τ_+ and τ_- are positive time constants defining the width of the positive and negative learning window. The resulting weight change at time t of synapse ji for a presynaptic spike train S_i^{pre} and a postsynaptic spike train S_j^{post} is usually modeled Gerstner and Kistler (2002) by the instantaneous application of this learning rule to all spike pairings with the second spike at time t

$$\left[\frac{d}{dt} w_{ji}(t)\right]_{STDP} = \int_0^\infty dr\ W(r) S_j^{post}(t) S_i^{pre}(t-r) + \int_0^\infty dr\ W(-r) S_j^{post}(t-r) S_i^{pre}(t). \tag{3.3}$$

The spike train of a neuron i which fires action potentials at times $t_i^{(1)}, t_i^{(2)}, t_i^{(3)}, \ldots$ is formalized here by a sum of Dirac delta functions $S_i(t) = \sum_n \delta(t - t_i^{(n)})$.

The model analyzed in this work is based on the assumption that positive and negative weight changes suggested by STDP for all pairs of pre- and postsynaptic spikes at synapse ji (according to the two integrals in (3.3)) are collected in an eligibility trace $c_{ji}(t)$ at the site of the synapse. The contribution to $c_{ij}(t)$ of all spike pairings with the second spike at time $t-s$ is modeled for $s>0$ by a function $f_c(s)$ (see Fig. 3.1A); the time scale of the eligibility trace is assumed in this work to be on the order of seconds. Hence the value of the eligibility trace of synapse ji at time t is given by

$$c_{ji}(t) = \int_0^\infty ds f_c(s) \left[\frac{d}{dt} w_{ji}(t-s)\right]_{STDP}, \tag{3.4}$$

see Fig. 3.1B. The actual weight change $\frac{d}{dt}w_{ji}(t)$ at time t for reward-modulated STDP is the product $c_{ij}(t)\cdot d(t)$ of the eligibility trace with the reward signal $d(t)$ as defined by equation (3.1). Since this simple model can in principle lead to unbounded growth of weights, we assume that weights are clipped at the lower boundary value 0 and an upper boundary w_{max}.

The network dynamics of a simulated recurrent network of spiking neurons where all connections between excitatory neurons are subject to STDP is quite sensitive to the particular STDP-rule that is used. Therefore we have carried out our network simulations not only with the additive STDP-rule (3.3), whose effect can be analyzed theoretically, but also with the more complex rule proposed in Morrison et al. (2007) (which was fitted to experimental data from hippocampal neurons in culture Bi and Poo (1998)), where the magnitude of the weight change depends on the current value of the weight. An implementation of this STDP-rule (with the parameters proposed in Morrison et al. (2007)) produced in our network simulations of the biofeedback experiment (computer simulation 1) as well as for learning pattern classification (computer simulation 4) qualitatively the same result as rule (3.3).

3.2.1 Theoretical analysis of the resulting weight changes

In this section, we derive a learning equation for reward-modulated STDP. This learning equation relates the change of a synaptic weight w_{ji} over some sufficiently long time interval T to statistical properties of the joint distribution of the reward signal $d(t)$ and pre- and postsynaptic firing times, under the assumption that the weight and correlations between pre- and postsynaptic spike times are slowly varying in time. We treat spike times as well as the reward signal $d(t)$ as stochastic variables. This mathematical framework allows us to derive the expected weight change over some time interval T (see Gerstner and Kistler (2002)), with the expectation taken over realizations of the stochastic input- and output spike trains as well as stochastic realizations of the reward signal, denoted by the ensemble average $\langle\cdot\rangle_E$

$$\frac{\langle w_{ji}(t+T) - w_{ji}(t)\rangle_E}{T} = \frac{1}{T}\left\langle \int_t^{t+T} \frac{d}{dt} w_{ji}(t')dt' \right\rangle_E = \left\langle \left\langle \frac{d}{dt} w_{ji}(t) \right\rangle_T \right\rangle_E, \tag{3.5}$$

where we used the abbreviation $\langle f(t)\rangle_T = T^{-1}\int_t^{t+T} f(t')\, dt'$. If synaptic plasticity is sufficiently slow, synaptic weights integrate a large number of small changes. In this case, the weight w_{ji} can be approximated by its average $\langle w_{ji}\rangle_E$ (it is “self-averaging”,

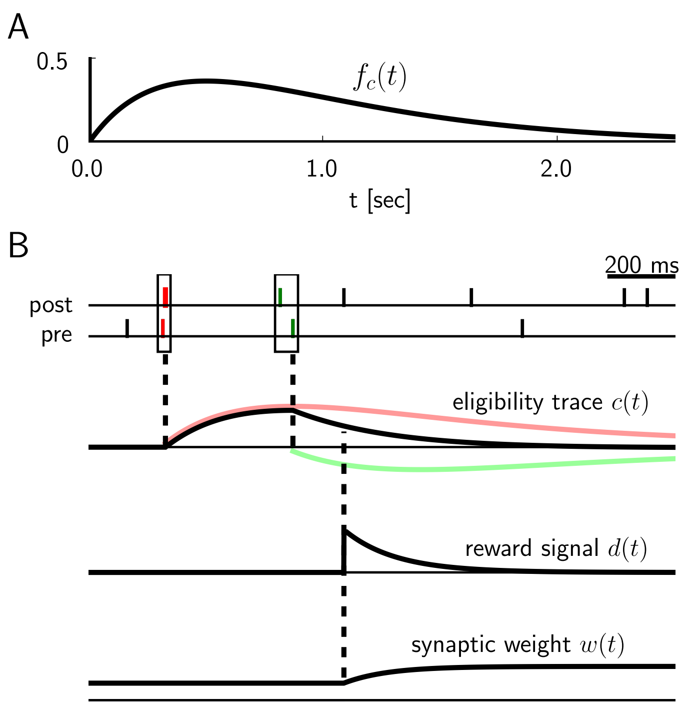

Figure 3.1: Scheme of reward-modulated STDP according to equations (3.1) - (3.4). **A)** Eligibility function $f_c(t)$, which scales the contribution of a pre/post spike pair (with the second spike at time 0) to the eligibility trace $c(t)$ at time t. **B)** Contribution of a pre-before-post spike pair (in red) and a post-before-pre spike pair (in green) to the eligibility trace $c(t)$ (in black), which is the sum of the red and green curves. According to equation (3.1) the change of the synaptic weight w is proportional to the product of $c(t)$ with a reward signal $d(t)$.

see Gerstner and Kistler (2002)). We can thus drop the expectation on the left hand side of equation (3.5) and write it as $\frac{d}{dt}\langle w_{ji}(t)\rangle_T$. Using equation (3.1), this yields (see Methods)

$$\boxed{\begin{aligned}\frac{d}{dt}\langle w_{ji}(t)\rangle_T &= \int_0^\infty dr\ W(r)\int_0^\infty ds\ f_c(s)\,\langle D_{ji}(t,s,r)\ \nu_{ji}(t-s,r)\rangle_T \\ &+\int_{-\infty}^0 dr\ W(r)\int_{|r|}^\infty ds\ f_c(s+r)\,\langle D_{ji}(t,s,r)\ \nu_{ji}(t-s,r)\rangle_T\,.\end{aligned}} \tag{3.6}$$

This formula contains the *reward correlation* for synapse ji

$$D_{ji}(t,s,r) = \langle d(t)|\text{ Neuron } j \text{ spikes at } t-s,\ \text{and neuron } i \text{ spikes at } t-s-r\rangle_E\ , \tag{3.7}$$

which is the average reward at time t given a presynaptic spike at time $t-s-r$ and a postsynaptic spike at time $t-s$. The joint firing rate $\nu_{ji}(t,r) = \langle S_j(t)S_i(t-$

$r)\rangle_E$ describes correlations between spike timings of neurons j and i, i.e., it is the probability density for the event that neuron i fires an action potential at time $t-r$ and neuron j fires an action potential at time t. For synapses subject to reward-modulated STDP, changes in efficacy are obviously driven by co-occurrences of spike pairings and rewards within the time scale of the eligibility trace. Equation (3.6) clarifies how the expected weight change depends on how the correlations between the pre- and postsynaptic neurons correlate with the reward signal.

If one assumes for simplicity that the impact of a spike pair on the eligibility trace is always triggered by the postsynaptic spike, one gets a simpler equation (see Methods)

$$\boxed{\frac{d}{dt}\left\langle w_{ji}(t)\right\rangle_T = \int_0^\infty ds\ f_c(s) \int_{-\infty}^{\infty} dr\ W(r) \left\langle D_{ji}(t,s,r)\ \nu_{ji}(t-s,r)\right\rangle_T .} \tag{3.8}$$

The assumption introduces a small error for post-before-pre spike pairs, because for a reward signal that arrives at some time d_r after the pairing, the weight update will be proportional to $f_c(d_r)$ instead of $f_c(d_r+r)$. The approximation is justified if the temporal average is performed on a much longer time scale than the time scale of the learning window, the effect of each pre-post spike pair on the reward signal is delayed by an amount greater than the time scale of the learning window, and f_c changes slowly compared to the time scale of the learning window (see Methods for details). For the analyzes presented in this work, the simplified equation (3.8) is a good approximation for the learning dynamics. Equation (3.8) is a generalized version of the STDP learning equation $\frac{d}{dt}w_{ji}(t) = \int_{-\infty}^{\infty} dr\ W(r) \left\langle \nu_{ji}(t-s,r)\right\rangle_T$ in Gerstner and Kistler (2002) that includes the impact of the reward correlation weighted by the eligibility function. To see the relation between standard STDP and reward-modulated STDP, consider a constant reward signal $d(t) = d_0$. Then also the reward correlation is constant and given by $D(t,s,r) = d_0$. We recover the standard STDP learning equation scaled by d_0 if the eligibility function is an instantaneous delta-pulse $f_c(s) = \delta(s)$. Furthermore, if the statistics of the reward signal $d(t)$ is time-independent and independent from the pre- and postsynaptic spike statistics of some synapse ji, then the reward correlation is given by $D_{ji}(t,s,r) = \langle d(t)\rangle_E = d_0$ for some constant d_0. Then, the weight change for synapse ji is $\frac{d}{dt}\langle w_{ji}(t)\rangle_T = d_0 \int_{-\infty}^{\infty} dr\ W(r) \int_0^\infty ds f_c(s) \left\langle \nu_{ji}(t-s,r)\right\rangle_T$. The temporal average of the joint firing rate $\left\langle \nu_{ji}(t-s,r)\right\rangle_T$ is thus filtered by the eligibility trace. We assumed in the preceding analysis that the temporal average is taken over some long time interval T. If the time scale of the eligibility trace is much smaller than this time interval T, then the weight change is approximately $\frac{d}{dt}\langle w_{ji}(t)\rangle_T \approx d_0(\int_0^\infty ds f_c(s)) \int_{-\infty}^{\infty} dr\ W(r) \left\langle \nu_{ji}(t,r)\right\rangle_T$, and the weight w_{ji} will change according to standard STDP scaled by a constant proportional to the mean reward and the integral over the eligibility function. In the remainder of this chapter, we will always use the smooth time-averaged weight change $\frac{d}{dt}\ \langle w_{ji}(t)\rangle_T$, but for brevity, we will drop the angular brackets and simply write $\frac{d}{dt}w_{ji}(t)$.

The learning equation (3.8) provides the mathematical basis for our following analyses. It allows us to determine synaptic weight changes if we can describe a learning situation in terms of reward correlations and correlations between pre- and postsynaptic spikes.

3.2.2 Application to models for biofeedback experiments

We now apply the preceding analysis to the biofeedback experiment of Fetz and Baker (1973) that were described in the introduction. These experiments pose the challenge to explain how learning mechanisms in the brain can detect and exploit correlations between rewards and the firing activity of one or a few neurons within a large recurrent network of neurons (the credit assignment problem), without changing the overall function or dynamics of the circuit.

We show that this phenomenon can in principle be explained by reward-modulated STDP. In order to do that, we define a model for the experiment which allows us to formulate an equation for the reward signal $d(t)$. This enables us to calculate synaptic weight changes for this particular scenario. We consider as model a recurrent neural circuit where the spiking activity of one neuron k is recorded by the experimenter.[2] We assume that in the monkey brain a reward signal $d(t)$ is produced which depends on the visual feedback (through an illuminated meter, whose pointer deflection was dependent on the current firing rate of the randomly selected neuron k) as well as previously received liquid rewards, and that this signal $d(t)$ is delivered to *all* synapses in large areas of the brain. We can formalize this scenario by defining a reward signal which depends on the spike rate of the arbitrarily selected neuron k (see Fig. 3.2A, B). More precisely, a reward pulse of shape $\varepsilon_r(r)$ (the reward kernel) is produced with some delay d_r every time the neuron k produces an action potential

$$d(t) = \int_0^\infty dr\ S_k^{post}(t - d_r - r)\varepsilon_r(r). \tag{3.9}$$

Note that $d(t) = h(t) - \bar{h}$ is defined in equation (3.1) as a signal with zero mean. In order to satisfy this constraint, we assume that the reward kernel ε_r has zero mass, i.e., $\bar{\varepsilon}_r = \int_0^\infty dr\ \varepsilon_r(r) = 0$. For the analysis, we use the linear Poisson neuron model described in Methods. The mean weight change for synapses to the reinforced neuron k is then approximately (see Methods)

$$\boxed{\frac{d}{dt}w_{ki}(t) \approx \int_0^\infty ds\ f_c(s + d_r)\varepsilon_r(s) \int_{-\infty}^\infty dr\ W(r) \left\langle \nu_{ki}(t - d_r - s, r)\right\rangle_T .} \tag{3.10}$$

This equation describes STDP with a learning rate proportional to $\int_0^\infty ds\ f_c(s + d_r)\varepsilon_r(s)$. The outcome of the learning session will strongly depend on this integral and thus on the form of the reward kernel ε_r. In order to reinforce high firing rates of the reinforced neuron we have chosen a reward kernel with a positive bump in

[2]Experiments where two neurons are recorded and reinforced were also reported in Fetz and Baker (1973). We tested this case in computer simulations (see Fig. (3.4)) but did not treat it explicitly in our theoretical analysis.

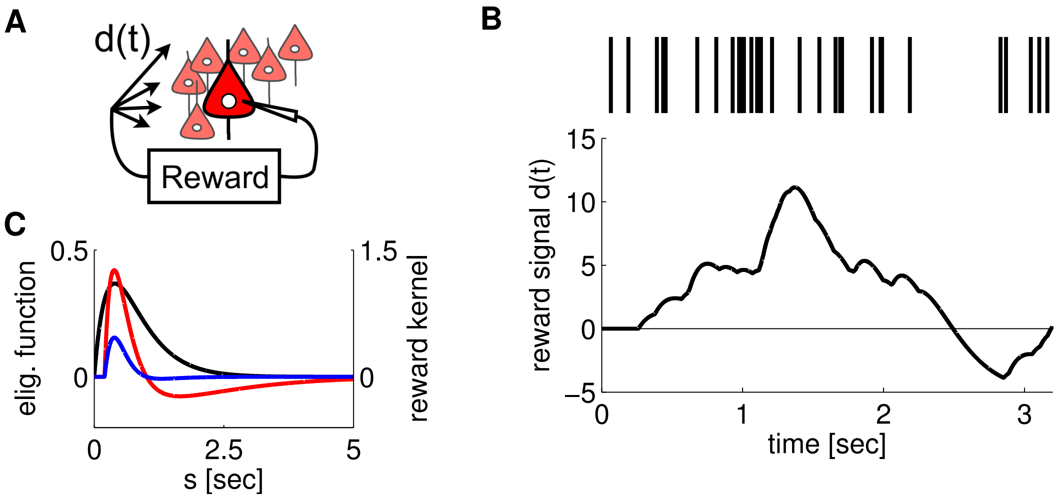

Figure 3.2: Setup of the model for the experiment by Fetz and Baker Fetz and Baker (1973). **A)** Schema of the model: The activity of a single neuron in the circuit determines the amount of reward delivered to all synapses between excitatory neurons in the circuit. **B)** The reward signal $d(t)$ in response to a spike train (shown at the top) of the arbitrarily selected neuron (which was selected from a recurrently connected circuit consisting of 4000 neurons). The level of the reward signal $d(t)$ follows the firing rate of the spike train. **C)** The eligibility function $f_c(s)$ (black curve, left axis), the reward kernel $\varepsilon_r(s)$ delayed by 200 ms (red curve, right axis), and the product of these two functions (blue curve, right axis) as used in our computer experiment. The integral of $f_c(s+d_r)\varepsilon_r(s)$ is positive, as required according to equation (3.10) in order to achieve a positive learning rate for the synapses to the selected neuron.

the first few hundred milliseconds, and a long negative tail afterwards. Fig. 3.2C shows the functions f_c and ε_r that were used in our computer model, as well as the product of these two functions. One sees that the integral over the product is positive and according to equation (3.10) the synapses to the reinforced neuron are subject to STDP.

This does not guarantee an increase of the firing rate of the reinforced neuron. Instead, the changes of neuronal firing will depend on the statistics of the inputs. In particular, the weights of synapses to neuron k will not increase if that neuron does not fire spontaneously. For uncorrelated Poisson input spike trains of equal rate, the firing rate of a neuron trained by STDP stabilizes at some value which depends on the input rate (see Song et al. (2000); Kempter et al. (2001)). However, in comparison to the low spontaneous firing rates observed in the biofeedback experiment Fetz and Baker (1973), the stable firing rate under STDP can be much higher, allowing for a significant rate increase. It was shown in Fetz and Baker (1973) that also low firing rates of a single neuron can be reinforced. In order to model this, we have chosen a reward kernel with a negative bump in the first few hundred milliseconds, and a long positive tail afterwards, i.e. we inverted the kernel used above to obtain a negative integral $\int_0^\infty ds\ f_c(s+d_r)\varepsilon_r(s)$. According to equation (3.10) this leads to anti-STDP where not only inputs to the reinforced neuron which have low correlations with the output are depressed (because of the negative integral of the learning window), but also those which are causally correlated with the output. This leads to a quick firing rate decrease at the reinforced neuron.

The mean weight change of synapses to non-reinforced neurons $j \neq k$ is given by

$$\boxed{\begin{aligned} \frac{d}{dt} w_{ji}(t) \approx & \int_0^\infty ds\ f_c(s) \int_{-\infty}^\infty dr\ W(r) \int_0^\infty dr' \varepsilon_r(r') \\ & \left\langle \frac{\nu_{kj}(t - d_r - r', s - d_r - r')}{\nu_j(t-s)} \nu_{ji}(t-s,r) \right\rangle_T , \end{aligned}} \tag{3.11}$$

where $\nu_j(t) = \langle S_j(t) \rangle_E$ is the instantaneous firing rate of neuron j at time t. This equation indicates that a non-reinforced neuron is trained by STDP with a learning rate proportional to its correlation with the reinforced neuron given by $\nu_{kj}(t - d_r - r', s - d_r - r')/\nu_j(t-s)$. In fact, it was noted in Fetz and Baker (1973) that neurons nearby the reinforced neuron tended to change their firing rate in the same direction. This observation might be explained by putative correlations of the recorded neuron with nearby neurons. On the other hand, if a neuron j is uncorrelated with the reinforced neuron k, we can decompose the joint firing rate into $\nu_{kj}(t - d_r - r', s - d_r - r') = \nu_k(t - d_r - r')\nu_j(t-s)$. In this case, the learning rate for synapse ji is approximately zero (see Methods). This ensures that most neurons in the circuit keep a constant firing rate, in spite of continuous weight changes according to reward-modulated STDP.

Altogether we see that the weights of synapses to the reinforced neuron k can only change if there is spontaneous activity in the network, so that in particular also this neuron k fires spontaneously. On the other hand the spontaneous network activity should not consist of repeating large-scale spatio-temporal firing patterns, since that would entail correlations between the firing of neuron k and other neurons j, and would lead to similar changes of synapses to these other neurons j. Apart from these requirements on the spontaneous network activity, the preceding theoretical results predict that stability of the circuit is preserved, while the neuron which is causally related to the reward signal is trained by STDP, if $\int_0^\infty ds\ f_c(s + d_r)\varepsilon_r(s)$ is positive.

3.2.2.1 Computer simulation 1: Model for biofeedback experiment

We tested these theoretical predictions through computer simulations of a generic cortical microcircuit receiving a reward signal which depends on the firing of one arbitrarily chosen neuron k from the circuit (reinforced neuron). The circuit was composed of 4000 LIF neurons, with 3200 being excitatory and 800 inhibitory, interconnected randomly by 228954 conductance based synapses with short term dynamics [3]. In addition to the explicitly modeled synaptic connections, conductance noise

[3] All computer simulations were also carried out as a control with static current based synapses, see Methods and Suppl.

(generated by an Ornstein-Uhlenbeck process) was injected into each neuron according to data from Destexhe et al. (2001), in order to model synaptic background activity of neocortical neurons in-vivo.[4] This background noise elicited spontaneous firing in the circuit at about 4.6 Hz. Reward-modulated STDP was applied continuously to all synapses which had excitatory presynaptic and postsynaptic neurons, and all these synapses received the same reward signal. The reward signal was modeled according to equation (3.9). Fig. 3.2C shows one reward pulse caused by a single postsynaptic spike at time $t = 0$ with the parameters used in the experiment. For several postsynaptic spikes, the amplitude of the reward signal follows the firing rate of the reinforced neuron, see Fig. 3.2B.

This model was simulated for 20 minutes of biological time. Panels A, B, D of Fig. 3.3 show that the firing rate of the reinforced neuron increases within a few minutes (like in the experiment of Fetz and Baker (1973)), while the firing rates of the other neurons remain largely unchanged. The increase of weights to the reinforced neuron shown in Fig. 3.3C can be explained by the correlations between its presynaptic and postsynaptic spikes shown in panel E. This panel shows that pre-before-post spike pairings (black curve) are in general more frequent than post-before-pre spike pairings. The reinforced neuron increases its rate from around 4 Hz to 12 Hz, which is comparable to the measured firing rates in Fetz and Baker (1973) before and after learning.

In Fig. 3.9 of Fetz and Baker (1973) and Fig. 3.1 of Fetz (2007) the results of another experiment were reported where the activity of two adjacent neurons was recorded, and high firing rates of the first neuron and low firing rates of the second neuron were reinforced simultaneously. This kind of differential reinforcement resulted in an increase and decrease of the firing rates of the two neurons correspondingly. We implemented this type of reinforcement by letting the reward signal in our model depend on the spikes of the two randomly chosen neurons (we refer to these neurons as neuron A and neuron B), i.e. $d(t) = d_+^A(t) + d_-^B(t)$, where $d_+^A(t)$ is the component that positively rewards spikes of neuron A, and $d_-^B(t)$ negatively rewards spikes of neuron B. Both parts of the reward signal, $d_+^A(t)$ and $d_-^B(t)$, were defined as in equation (3.9) for the corresponding neuron. For $d_+^A(t)$ we used the reward kernel ε_r as defined in equation (3.29), whereas for $d_-^B(t)$ we used $\varepsilon_{r-} = -\varepsilon_r$ (note that the integral over ε_{r-} is still zero). At the middle of the simulation (simulation time $t = 10$min), we changed the direction of the reinforcements by negatively rewarding the firing of neuron A and positively rewarding the firing of neuron B (i.e., $d(t) = d_-^A(t) + d_+^B(t)$). The results are summarized in Fig. 3.4. With a reward signal modeled in this way, we were able to independently increase and decrease the firing rates of the two neurons according to the reinforcements, while the firing rates of the other neurons remained unchanged. Changing the type of reinforcement during the

[4]More precisely, for 50% of the excitatory neurons the amplitude of the noise injection was reduced to 20%, and instead their connection probabilities from other excitatory neurons were chosen to be larger (see Methods and Fig. S1 and S2 for details). The reinforced neuron had to be chosen from the latter population, since reward-modulated STDP does not work properly if the postsynaptic neuron fires too often because of directly injected noise.

simulation from positive to negative for neuron A and from negative to positive for neuron B resulted in a corresponding shift in their firing rate change in the direction of the reinforcement.

The dynamics of a network where STDP is applied to all synapses between excitatory neurons is quite sensitive to the specific choice of the STDP-rule. The preceding theoretical analysis (see equation (3.10), (3.11)) predicts that reward-modulated STDP affects in the long run only those excitatory synapses where the firing of the postsynaptic neuron is correlated with the reward signal. In other words: the reward signal gates the effect of STDP in a recurrent network, and thereby can keep the network within a given dynamic regime. This prediction is confirmed qualitatively by the two panels of Fig. 3.3A, which show that even after all excitatory synapses in the recurrent network have been subject to 20 minutes (in simulated biological time) of reward-modulated STDP, the network stays within the asynchronous irregular firing regime. It is also confirmed quantitatively through Fig. 3.5. These figures show results for the simple additive version of STDP (according to equation (3.3)). Very similar results (see Fig. S3 and S4) arise from an application of the more complex STDP-rule proposed in Morrison et al. (2007) where the weight-change depends on the current weight value.

3.2.3 Rewarding spike-times

The preceding model for the biofeedback experiment of Fetz and Baker focused on learning of firing rates. In order to explore the capabilities and limitations of reward-modulated STDP in contexts where the temporal structure of spike trains matters, we investigated another reinforcement learning scenario where a neuron should learn to respond with particular temporal spike patterns. We first apply analytical methods to derive conditions under which a neuron subject to reward-modulated STDP can achieve this.

In this model, the reward signal $d(t)$ is given in dependence on how well the output spike train S_j^{post} of a neuron j matches some rather arbitrary spike train S^* (which might for example represent spike output from some other brain structure during a developmental phase). S^* is produced by a neuron μ^* that receives the same n input spike trains $S_1, \ldots, S_n$ as the trained neuron j, with some arbitrarily chosen weights $\mathbf{w}^* = (w_1^*, \ldots, w_n^*)^T$, $w_i^* \in \{0, w_{max}\}$. But in addition the neuron μ^* receives $n' - n$ further spike trains $S_{n+1}, \ldots, S_{n'}$ with weights $w_{n+1}^*, \ldots, w_{n'}^* = w_{max}$. The setup is illustrated in Fig. 3.6A. It provides a generic reinforcement learning scenario, when a quite arbitrary (and not perfectly realizable) spike output is reinforced, but simultaneously the performance of the learner can be evaluated clearly according to how well its weights $w_{j1}, \ldots, w_{jn}$ match those of the neuron μ^* for those n input spike trains which both of them have in common. The reward $d(t)$ at time t depends in this task on both the timing of action potentials of the trained neuron and spike times in the target spike train S^*

$$d(t) = \int_{-\infty}^{\infty} dr \ \kappa(r) S_j^{post}(t - d_r) S^*(t - d_r - r), \tag{3.12}$$

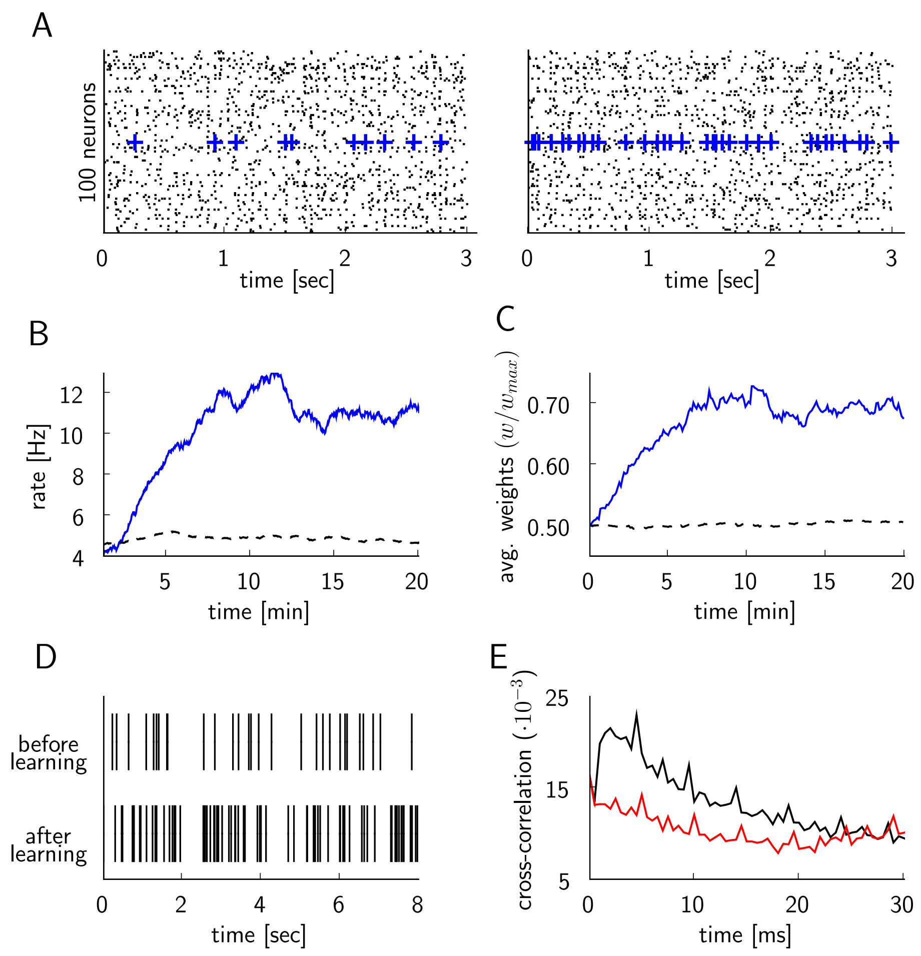

Figure 3.3: Simulation of the experiment by Fetz and Baker Fetz and Baker (1973) for the case where an arbitrarily selected neuron triggers global rewards when it increases its firing rate. **A)** Spike response of 100 randomly chosen neurons within the recurrent network of 4000 neurons at the beginning of the simulation (20sec - 23sec, left plot), and at the end of the simulation (the last 3 seconds, right plot). The firing times of the reinforced neuron are marked by blue crosses. **B)** The firing rate of the positively rewarded neuron (blue line) increases, while the average firing rate of 20 other randomly chosen neurons (dashed line) remains unchanged. **C)** Evolution of the average weight of excitatory synapses to the reinforced neuron (blue line), and of the average weight of 1663 randomly chosen excitatory synapses to other neurons in the circuit (dashed line). **D)** Spike trains of the reinforced neuron before and after learning. **E)** Histogram of the time-differences between presynaptic and postsynaptic spikes (bin size 0.5ms), averaged over all excitatory synapses to the reinforced neuron. The black curve represents the histogram values for positive time differences (when the presynaptic spike precedes the postsynaptic spike), and the red curve represents the histogram for negative time differences.

where the function $\kappa(r)$ with $\bar{\kappa} = \int_{-\infty}^{\infty} ds\ \kappa(s) > 0$ describes how the reward signal depends on the time difference r between a postsynaptic spike and a target spike,

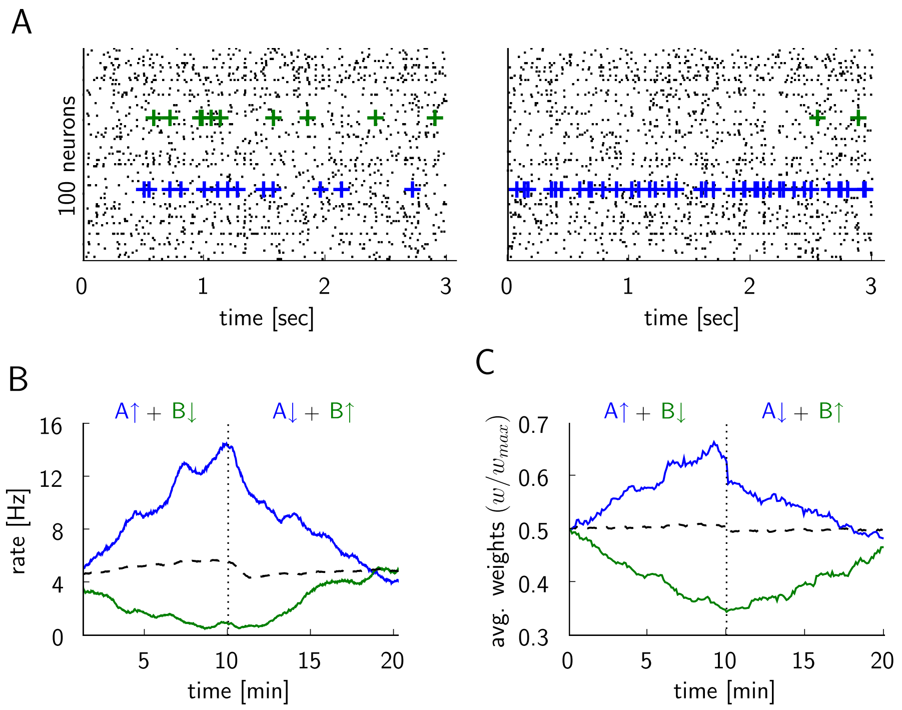

Figure 3.4: Differential reinforcement of two neurons (within a simulated network of 4000 neurons, the two rewarded neurons are denoted as A and B), corresponding to the experimental results shown in Fig. 3.9 of Fetz and Baker (1973) and Fig. 3.1 of Fetz (2007). **A)** The spike response of 100 randomly chosen neurons at the beginning of the simulation (20sec - 23sec, left plot), and at the middle of simulation just before the switching of the reward policy (597sec-600sec, right plot). The firing times of the first reinforced neuron A are marked by blue crosses and those of the second reinforced neuron B are marked by green crosses. **B)** The dashed vertical line marks the switch of the reinforcements at $t = 10$min. The firing rate of neuron A (blue line) increases while it is positively reinforced in the first half of the simulation and decreases in the second half when its spiking is negatively reinforced. The firing rate of the neuron B (green line) decreases during the negative reinforcement in the first half and increases during the positive reinforcement in the second half of the simulation. The average firing rate of 20 other randomly chosen neurons (dashed line) remains unchanged. **C)** Evolution of the average weight of excitatory synapses to the rewarded neurons A and B (blue and green lines respectively), and of the average weight of 1744 randomly chosen excitatory synapses to other neurons in the circuit (dashed line).

and $d_r > 0$ is the delay of the reward.

Our theoretical analysis (see Methods) predicts that under the assumption of constant-rate uncorrelated Poisson input statistics this reinforcement learning task can be solved by reward-modulated STDP for arbitrary initial weights if three constraints are fulfilled:

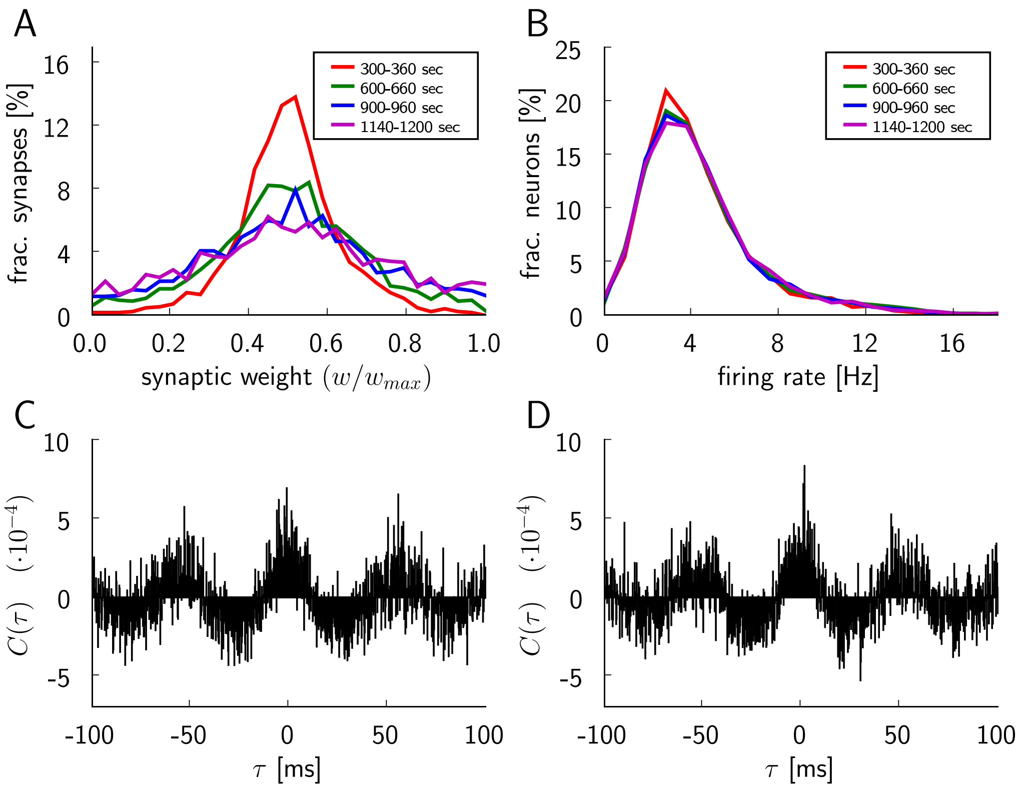

Figure 3.5: Evolution of the dynamics of a recurrent network of 4000 LIF neurons during application of reward-modulated STDP. **A)** Distribution of the synaptic weights of excitatory synapses to 50 randomly chosen non-reinforced neurons, plotted for 4 different periods of simulated biological time during the simulation. The weights are averaged over 10 samples within these periods. The colors of the curves and the corresponding intervals are as follows: red (300 − 360 sec), green (600 − 660 sec), blue (900 − 960 sec), magenta (1140 − 1200 sec). **B)** The distribution of average firing rates of the non-reinforced excitatory neurons in the circuit, plotted for the same time periods as in A). The colors of the curves are the same as in A). The distribution of the firing rates of the neurons in the circuit remains unchanged during the simulation, which covers 20 minutes of biological time. **C)** Cross-correlogram of the spiking activity in the circuit, averaged over 200 pairs of non-reinforced neurons and over 60 s, with a bin size of 0.2 ms, for the period between 300 and 360 seconds of simulated biological time. It is calculated as the cross-covariance divided by the square root of the product of variances. **D)** As in C), but between seconds 1140 and 1200.(Separate plots of panel B, C, D for two types of excitatory neurons that received different amounts of noise currents are given in Fig. S1 and S2.)

$$\boxed{\begin{aligned} -\nu_{min}^{post}\bar{W} &> w_{max}\bar{W}_{\varepsilon} && (3.13)\\ \int_{-\infty}^{\infty} dr\ W(r)\varepsilon(r)\varepsilon_{\kappa}(r) &\geq -\nu_{max}^{post}\bar{W}\int_{0}^{\infty} dr\ \varepsilon(r)\varepsilon_{\kappa}(r) && (3.14)\\ \int_{-\infty}^{\infty} dr\ W(r)\varepsilon_{\kappa}(r) &> -\bar{W}\bar{\kappa}\left[\frac{\nu^{*}\nu_{max}^{post}}{w_{max}}\frac{\bar{f}_c}{f_c(d_r)}+\frac{\nu^{*}}{w_{max}}+\nu^{*}+\nu_{max}^{post}\right] && (3.15)\end{aligned}}$$

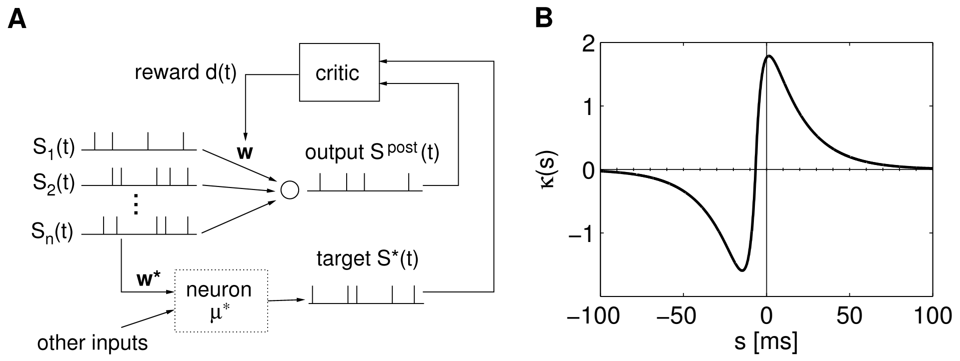

Figure 3.6: Setup for reinforcement learning of spike times. **A)** Architecture. The trained neuron receives n input spike trains. The neuron μ^* receives the same inputs plus additional inputs not accessible to the trained neuron. The reward is determined by the timing differences between the action potentials of the trained neuron and the neuron μ^*. **B)** A reward kernel with optimal offset from the origin of $t_\kappa = -6.6$ms. The optimal offset for this kernel was calculated with respect to the parameters from computer simulation 1 in Table 3.1. Reward is positive if the neuron spikes around the target spike or somewhat later, and negative if the neuron spikes much too early.

The following parameters occur in these equations: ν^* is the output rate of neuron μ^*, ν_{min}^{post} is the minimal output rate, ν_{max}^{post} is the maximal output rate of the trained neuron, $\bar{f}_c = \int_0^\infty dr\ f_c(r)$ is the integral over the eligibility trace, $\bar{W} = \int_{-\infty}^\infty dr\ W(r)$ is the integral over the STDP learning curve (see equation (3.2)), $\varepsilon_\kappa(r) = \int_{-\infty}^\infty dr'\ \kappa(r')\varepsilon(r-r')$ is the convolution of the reward kernel with the shape of the postsynaptic potential (PSP) $\varepsilon(s)$, and $\bar{W}_\varepsilon = \int_{-\infty}^\infty dr\ \varepsilon(r)W(r)$ is the integral over the PSP weighted by the learning window.

If these inequalities are fulfilled and input rates are larger than zero, then the weight vector of the trained neuron converges on average from any initial weight vector to $\mathbf{w}^*$ (i.e., it mimics the weight distribution of neuron μ^* for those n inputs which both have in common). To get an intuitive understanding of these inequalities, we first examine the idea behind constraint (3.13). This constraint assures that weights of synapses i with $w_i^* = 0$ decay to zero in expectation. First note that input spikes from a spike train S_i with $w_i^* = 0$ have no influence on the target spike train S^*. In the linear Poisson neuron model, this leads to weight changes similar to STDP which can be described by two terms. First, all synapses are subject to depression stemming from the negative part of the learning curve W and random pre-post spike pairs. This weight change is bounded from below by $\alpha\nu_i^{pre}\nu_{min}^{post}\bar{W}$ for some positive constant α. On the other hand, the positive influence of input spikes on postsynaptic firing leads to potentiation of the synapse bounded from above by $\alpha\nu_i^{pre}w_{max}\bar{W}_\varepsilon$. Hence the weight decays to zero if $-\alpha\nu_i^{pre}\nu_{min}^{post}\bar{W} > \alpha\nu_i^{pre}w_{max}\bar{W}_\varepsilon$, leading to inequality (3.13). For synapses i with $w_i^* = w_{max}$, there is an additional drive, since each presynaptic spike increases the probability of a closely following spike in the target spike train S^*. Therefore, the probability of a delayed reward signal after a presynaptic spike is larger. This additional drive leads to positive weight changes if inequalities (3.14) and (3.15) are fulfilled (see Methods).

Note that also for the learning of spike times spontaneous spikes (which might be regarded as "noise") are important, since they may lead to reward signals that can be exploited by the learning rule. It is obvious that in reward-modulated STDP, a silent neuron cannot recover from its silent state, since there will be no spikes which can drive STDP. But in addition, condition (3.13) shows that in this learning scenario, the minimal output rate ν_{min}^{post} – which increases with increasing noise – has to be larger than some positive constant, such that depression is strong enough to weaken synapses if needed. On the other hand, if the noise is too strong also synapses i with $w_i = w_{max}$ will be depressed and may not converge correctly. This can happen when the increased noise leads to a maximal postsynaptic rate ν_{max}^{post} such that constraints (3.14) and (3.15) are not satisfied anymore.

The conditions (3.13)-(3.15) also reveal how parameters of the model influence the applicability of this setup. For example, the eligibility trace enters the equations only in the form of its integral and its value at the reward delay in equation (3.15). In fact, the exact shape of the eligibility trace is not important. The important property of an ideal eligibility trace is that it is high at the reward delay and low at other times as expressed by the fraction in condition (3.15). Interestingly, the formulas also show that one has quite some freedom in choosing the form of the STDP window, as long as the reward kernel ε_κ is adjusted accordingly. For example, instead of a standard STDP learning window W with $W(r) \geq 0$ for $r > 0$ and $W(r) \leq 0$ for $r < 0$ and a corresponding reward kernel κ, one can use a reversed learning window W' defined by $W'(r) \equiv W(-r)$ and a reward kernel κ' such that $\varepsilon_{\kappa'}(r) = \varepsilon_\kappa(-r)$. If (3.15) is satisfied for W and κ, then it is also satisfied for W' and κ' (and in most cases also condition (3.14) will be satisfied). This reflects the fact that in reward modulated STDP the learning window defines the weight changes in combination with the reward signal.

For a given STDP learning window, the analysis reveals what reward kernels κ are suitable for this learning setup. From condition (3.15), we can deduce that the integral over κ should be small (but positive), whereas the integral $\int_{-\infty}^{\infty} dr\, W(r)\varepsilon_\kappa(r)$ should be large. Hence, for a standard STDP learning window W with $W(r) \geq 0$ for $r > 0$ and $W(r) \leq 0$ for $r < 0$, the convolution $\varepsilon_\kappa(r)$ of the reward kernel with the PSP should be positive for $r > 0$ and negative for $r < 0$. In the computer simulation we used a simple kernel depicted in Fig. 3.6B, which satisfies the aforementioned constraints. It consists of two double-exponential functions, one positive and one negative, with a zero crossing at some offset t_κ from the origin. The optimal offset t_κ is always negative and in the order of several milliseconds for usual PSP-shapes ε. We conclude that for successful learning in this scenario, a positive reward should be produced if the neuron spikes around the target spike or somewhat later, and a negative reward should be produced if the neuron spikes much too early.

3.2.3.1 Computer simulation 2: Learning spike times

In order to explore this learning scenario in a biologically more realistic setting, we trained a LIF neuron with conductance based synapses exhibiting short term facilitation and depression. The trained neuron and the neuron μ^* which produced the target spike train S^* both received inputs from 100 input neurons emitting spikes from a constant rate Poisson process of 15 Hz. The synapses to the trained neuron were subject to reward-modulated STDP. The weights of neuron μ^* were set to $w_i^* = w_{max}$ for $0 \leq i < 50$ and $w_i^* = 0$ for $50 \leq i < 100$. In order to simulate a non-realizable target response, neuron μ^* received 10 additional synaptic inputs (with weights set to $w_{max}/2$). During the simulations we observed a firing rate of 18.2 Hz for the trained neuron, and 25.2 Hz for the neuron μ^*. The simulations were run for 2 hours simulated biological time.

We performed 5 repetitions of the experiment, each time with different randomly generated inputs and different initial weight values for the trained neuron. In each of the 5 runs, the average synaptic weights of synapses with $w_i^* = w_{max}$ and $w_i^* = 0$ approached their target values, as shown in Fig. 3.7A. In order to test how closely the trained neuron reproduces the target spike train S^* after learning, we performed additional simulations where the same spike input was applied to the trained neuron before and after the learning. Then we compared the output of the trained neuron before and after learning with the output S^* of neuron μ^*. Fig. 3.7B shows that the trained neuron approximates the part of S^* which is accessible to it quite well. Panels C-F of Fig. 3.7 provide more detailed analyses of the evolution of weights during learning. The computer simulations confirmed the theoretical prediction that the neuron can learn well through reward-modulated STDP only if a certain level of noise is injected into the neuron (see preceding discussion and the Fig. S6).

Both the theoretical results and these computer simulations demonstrate that a neuron can learn quite well through reward-modulated STDP to respond with specific spike patterns.

3.2.3.2 Computer simulation 3: Testing the analytically derived conditions

Equations (3.13) - (3.15) predict under which relationships between the parameters involved the learning of particular spike responses through reward-modulated STDP will be successful. We have tested these predictions by selecting 6 arbitrary settings of these parameters, which are listed in Table 3.1. In 4 cases (marked by light gray shading in Fig. 3.8) these conditions were not met (either for the learning of weights with target value w_{max}, or for the learning of weights with target value 0. Fig. 3.8 shows that the derived learning result is not achieved in exactly these 4 cases. On the other hand, the theoretically predicted weight changes (black bar) predict in all cases the actual weight changes (gray bar) that occur for the chosen simulation times (listed in the last column of Table 3.1) remarkably well.

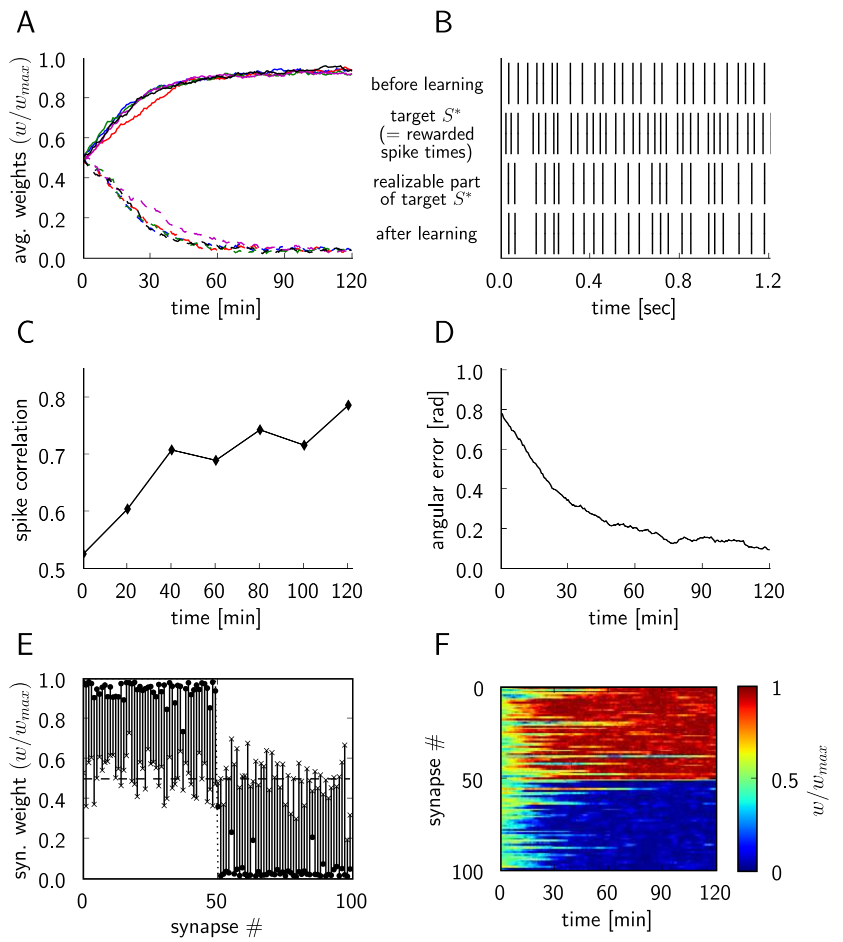

Figure 3.7: Results for reinforcement learning of exact spike times through reward-modulated STDP. **A)** Synaptic weight changes of the trained LIF neuron, for 5 different runs of the experiment. The curves show the average of the synaptic weights that should converge to $w_i^* = 0$ (dashed lines), and the average of the synaptic weights that should converge to $w_i^* = w_{max}$ (solid lines) with different colors for each simulation run. **B)** Comparison of the output of the trained neuron before (top trace) and after learning (bottom trace). The same input spike trains and the same noise inputs were used before and after training for 2 hours. The second trace from above shows those spike times S^* which are rewarded, the third trace shows the realizable part of S^* (i.e. those spikes which the trained neuron could potentially learn to reproduce, since the neuron μ^* produces them without its 10 extra spike inputs). The close match between the third and fourth trace shows that the trained neuron performs very well. **C)** Evolution of the spike correlation between the spike train of the trained neuron and the realizable part of the target spike train S^*. **D)** The angle between the weight vector $\mathbf{w}$ of the trained neuron and the weight vector $\mathbf{w}^*$ of the neuron μ^* during the simulation, in radians. **E)** Synaptic weights at the beginning of the simulation are marked with $\times$, and at the end of the simulation with $\bullet$, for each plastic synapse of the trained neuron. **F)** Evolution of the synaptic weights w/w_{max} during the simulation (we had chosen $w_i^* = w_{max}$ for $i < 50$, $w_i^* = 0$ for $i \geq 50$).

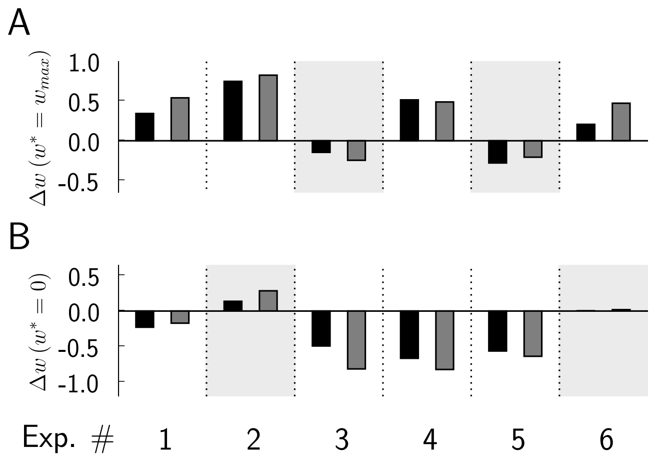

Figure 3.8: Test of the validity of the analytically derived conditions (3.13)-(3.15) on the relationship between parameters for successful learning with reward-modulated STDP. Predicted average weight changes (black bars) calculated from equation (3.22) match in sign and magnitude the actual average weight changes (gray bars) in computer simulations, for 6 different experiments with different parameter settings (see Table 3.1). **A)** Weight changes for synapses with $w_i^* = w_{max}$. **B)** Weight changes for synapses with $w_i^* = 0$. Four cases where the constraints (3.13) - (3.15) are not fulfilled are shaded in light gray. In all of these four cases the weights move into the opposite direction, i.e., a direction that decreases rewards.

Ex.	τ_ε[ms]	w_{max}	ν_{min}^{post} [Hz]	$A_+ 10^6$	$\frac{A_-}{A_+}$	τ_+ [ms]	A_+^κ, A_-^κ	τ_2^κ [ms]	t_{sim} [h]
1	10	0.012	10	16.62	1.05	20	3.34, -3.12	20	5
2	7	0.020	5	11.08	1.02	15	4.58, -4.17	16	10
3	20	0.010	6	5.54	1.10	25	1.50, -1.39	40	19
4	7	0.020	5	11.08	1.07	25	4.67, -4.17	16	13
5	10	0.015	6	20.77	1.10	25	3.75, -3.12	20	2
6	25	0.005	3	13.85	1.01	25	3.34, -3.12	20	18

Table 3.1: Parameter values used for computer simulation 3 (see Fig. 3.8).

3.2.4 Pattern discrimination with reward-modulated STDP

We examine here the question whether a neuron can learn through reward-modulated STDP to discriminate between two spike patterns P and N of its presynaptic neurons, by responding with more spikes to pattern P than to pattern N. Our analysis is based on the assumption that there exist internal rewards $d(t)$ that could guide such pattern discrimination. This reward based learning architecture is biologically more plausible than an architecture with a supervisor which provides for each input pattern a target output and thereby directly produces the desired firing behavior of the neuron (since the question becomes then how the supervisor has learnt to produce the desired spike outputs).

We consider a neuron that receives input from n presynaptic neurons. A pattern X consists of n spike trains, each of time length T, one for each presynaptic neuron. There are two patterns, P and N, which are presented in alternation to the neuron, with some reset time between presentations. For notational simplicity, we assume that each of the n presynaptic spike trains consists of exactly one spike. Hence, each pattern can be defined by a list of spike times: $P = (t_1^P, \dots, t_n^P)$, $N = (t_1^N, \dots, t_n^N)$, where t_i^X is the time when presynaptic neuron i spikes for pattern $X \in \{P, N\}$. A generalization to the easier case of learning to discriminate spatio-temporal presynaptic firing patterns (where some presynaptic neurons produce different numbers of spikes in different patterns) is straightforward, however the main characteristics of the learning dynamics are better accessible in this conceptually simpler setup. It had already been shown in Izhikevich (2007) that neurons can learn through reward-modulated STDP to discriminate between different *spatial* presynaptic firing patterns. But in the light of the analysis of Farries and Fairhall (2007) it is still open whether neurons can learn with simple forms of reward-modulated STDP, such as the one considered in this work, to discriminate *temporal* presynaptic firing patterns.

We assume that the reward signal $d(t)$ rewards – after some delay d_r – action potentials of the trained neuron if pattern P was presented, and punishes action potentials of the neuron if pattern N was presented. More precisely, we assume that

$$d(t) = \begin{cases} \alpha^P \int_0^\infty dr\ \varepsilon_r(r) S^{post}(t - d_r - r) & , \quad \text{if a pattern } P \text{ was presented} \\ \alpha^N \int_0^\infty dr\ \varepsilon_r(r) S^{post}(t - d_r - r) & , \quad \text{if a pattern } N \text{ was presented} \end{cases} \tag{3.16}$$

with some reward kernel ε_r and constants $\alpha^N < 0 < \alpha^P$. The goal of this learning task is to produce many output spikes for pattern P, and few or no spikes for pattern N.

The main result of our analysis is an estimate of the expected weight change of synapse i of the trained neuron for the presentation of pattern P, followed after a sufficiently long time T' by a presentation of pattern N

$$\Delta w_i = \int_0^{T'} dt \left[\left\langle \frac{dw_i(t)}{dt} \right\rangle_{E|P} + \left\langle \frac{dw_i(t)}{dt} \right\rangle_{E|N} \right],$$

where $\langle \cdot \rangle_{E|X}$ is the expectation over the ensemble given that pattern X was presented. This weight change can be estimated as (see Methods)

$$\boxed{\Delta w_i = \int_{-\infty}^{\infty} dr W(r) \left[\nu^P(t_i^P + r) A_i^P + \nu^N(t_i^N + r) A_i^N \right],} \tag{3.17}$$

where $\nu^X(t)$ is the postsynaptic rate at time t for pattern X, and the constants A_i^X for $X \in \{P, N\}$ are given by

$$A_i^X = \alpha^X \int_0^\infty dr' \varepsilon_r(r') \left[f_c(d_r + r') + \int_0^{T'} dt f_c(t - t_i^X) \nu^X(t - d_r - r') \right]. \tag{3.18}$$

As we will see shortly, an interesting learning effect is achieved if A_i^P is positive and A_i^N is negative. Since $f_c(r)$ is non-negative, a natural way to achieve this is to choose a positive reward kernel $\varepsilon_r(r) \geq 0$ for $r > 0$ and $\varepsilon_r(r) = 0$ for $r < 0$ (also, $f_c(r)$ and $\varepsilon_r(r)$ must not be identical to zero for all r).

We use equation (3.17) to provide insight on when and how the classification of temporal spike patterns can be learnt with reward-modulated STDP. Assume for the moment that $A_i^N = -A_i^P$. We first note that it is impossible to achieve through any synaptic plasticity rule that the time integral over the membrane potential of the trained neuron has after training a larger value for input pattern P than for input pattern N. The reason is that each presynaptic neuron emits the same number of spikes in both patterns (namely one spike). This simple fact implies that it is impossible to train a linear Poisson neuron (with any learning method) to respond to pattern P with more spikes than to pattern N. But equation (3.17) implies that reward-modulated STDP increases the variance of the membrane potential for pattern P, and reduces the variance for pattern N. This can be seen as follows. Because of the specific form of the STDP learning curve $W(r)$, which is positive for (small) positive r, negative for (small) negative r, and zero for large r, $\Delta w_i = \int_{-\infty}^{\infty} dr W(r) \nu^P(t_i^P + r) A_i^P$ has a potentiating effect on synapse i if the postsynaptic rate for pattern P is larger (because of a higher membrane potential) shortly after the presynaptic spike at this synapse i than before that spike. This tends to further increase the membrane potential after that spike. On the other hand, since A_i^N is negative, the same situation for pattern N has a depressing effect on synapse i, which counteracts the increased membrane potential after the presynaptic spike. Dually, if the postsynaptic rate shortly after the presynaptic spike at synapse i is lower than shortly before that spike, the effect on synapse i is depressing for pattern P. This leads to a further decrease of the membrane potential after that spike. In the same situation for pattern N, the effect is potentiating, again counteracting the variation of the membrane potential. The total effect on the postsynaptic membrane potential is that the fluctuations for pattern P are increased, while the membrane potential for pattern N is flattened.

For the LIF neuron model, and most reasonable other non-linear spiking neuron models, as well as for biological neurons in-vivo and in-vitro Stevens and Zador (1998); Mainen and Sejnowski (1995); Silberberg et al. (2004), larger fluctuations of the membrane potential lead to more action potentials. As a result, reward-modulated STDP tends to increase the number of spikes for pattern P for these neuron models, while it tends to decrease the number of spikes for pattern N, thereby enabling a discrimination of these purely temporal presynaptic spike patterns.

3.2.4.1 Computer simulation 4: Learning pattern classification

We tested these theoretical predictions through computer simulations of a LIF neuron with conductance based synapses exhibiting short-term depression and facilitation. Both patterns, P and N, had 200 input channels, with 1 spike per channel (hence this is the extreme where *all* information lies in the timing of presynaptic spikes). The spike times were drawn from an uniform distribution over a time interval of 500ms, which was the duration of the patterns. We performed 1000 training trials where the patterns P and N were presented to the neuron in alternation. To introduce exploration for this reinforcement learning task, the neuron had injected 20% of the Ornstein-Uhlenbeck process conductance noise (see Methods for further details).

The theoretical analysis predicted that the membrane potential will have after learning a higher variance for pattern P, and a lower variance for pattern N. When in our simulation of a LIF neuron the firing of the neuron was switched off (by setting the firing threshold potential too high) we could observe the membrane potential fluctuations undisturbed by the reset mechanism after each spike (see Fig. 3.9C, D). The variance of the membrane potential did in fact increase for pattern P from $2.49(mV)^2$ to $5.43(mV)^2$ (panel C), and decrease for pattern N (panel D), from $2.34(mV)^2$ to $1.33(mV)^2$. The corresponding plots with the firing threshold included are given in panels E and F, showing an increased member of spikes of the LIF neuron for pattern P, and a decreased number of spikes for pattern N. Furthermore, as panels A and B in Fig. 3.9 show, the increased variance of the membrane potential for the positively reinforced pattern P led to a stable temporal firing pattern in response to pattern P.

We repeated the experiment 6 times, each time with different randomly generated patterns P and N, and different random initial synaptic weights of the neuron. The results in Fig. 3.9 G and H show that the learning of temporal pattern discrimination through reward-modulated STDP does not depend on the temporal patterns that are chosen, nor on the initial values of synaptic weights.

3.2.4.2 Computer simulation 5: Training a readout neuron with reward-modulated STDP to recognize isolated spoken digits

A longstanding open problem is how a biologically realistic neuron model can be trained in a biologically plausible manner to extract information from a generic cortical microcircuit. Previous work Maass et al. (2002b, 2004); Destexhe and Marder (2004); Maass et al. (2007); Nikolić et al. (2007) has shown that quite a bit of salient information about recent and past inputs to the microcircuit can be extracted by a non-spiking linear readout neuron (i.e., a perceptron) that is trained by linear regression or margin maximization methods. Here we examine to what extent a LIF readout neuron with conductance based synapses (subject to biologically realistic short term synaptic plasticity) can learn through reward-modulated STDP to extract from the response of a simulated cortical microcircuit (consisting of 540 LIF neurons), see Fig. 3.10A, the information which spoken digit (transformed into spike trains by a standard cochlea model) is injected into the circuit. In comparison with the preceding task in simulation 4, this task is easier because the presynaptic firing patterns that need to be discriminated differ in temporal and spatial aspects (see Fig. 3.10B; Fig. S10 and 11 show the spike trains that were injected into the circuit). But this task is on the other hand more difficult, because the circuit re-

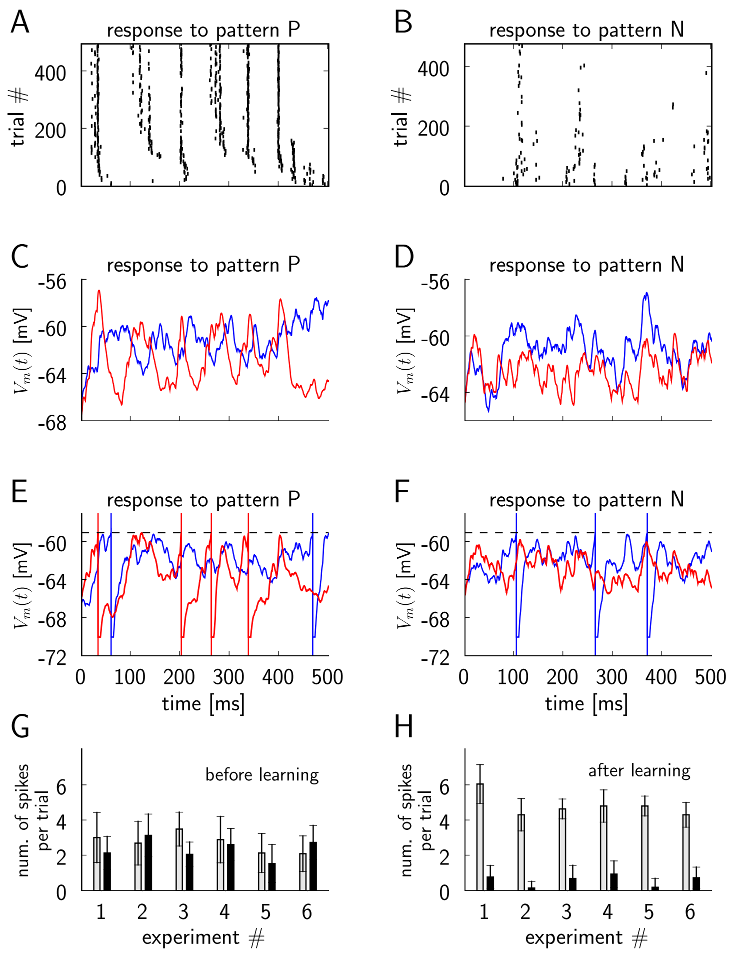

Figure 3.9: See next page for figure caption.

sponse (which creates the presynaptic firing pattern for the readout neuron) differs also significantly for two utterances of the same digit (Fig. 3.10C), and even for two trials for the same utterance (Fig. 3.10D) because of the intrinsic noise in the circuit (which was modeled according to Destexhe et al. (2001) to reflect in-vivo conditions during cortical UP-states). The results shown in Fig. 3.10E - H demonstrate that nevertheless this learning experiment was successful. On the other hand we were not able to achieve in this way speaker-independent word recognition, which had been achieved in Maass et al. (2002b) with a linear readout. Hence further work will be needed in order to clarify whether biologically more realistic models for readout neurons can be trained through reinforcement learning to reach the classification capabilities of perceptrons that are trained through supervised learning.

Figure 3.9: Training a LIF neuron to classify purely temporal presynaptic firing patterns: a positive reward is given for firing of the neuron in response to a temporal presynaptic firing pattern P, and a negative reward for firing in response to another temporal pattern N. **A)** The spike response of the neuron for individual trials, during 500 training trials when pattern P is presented. Only the spikes from every 4-th trial are plotted. **B)** As in A), but in response to pattern N. **C)** The membrane potential $V_m(t)$ of the neuron during a trial where pattern P is presented, before (blue curve) and after training (red curve), with the firing threshold removed. The variance of the membrane potential increases during learning, as predicted by the theory. **D)** As in C), but for pattern N. The variance of the membrane potential for pattern N decreases during learning, as predicted by the theory. **E)** The membrane potential $V_m(t)$ of the neuron (including action potentials) during a trial where pattern P is presented before (blue curve) and after training (red curve). The number of spikes increases. **F)** As in E), but for trials where pattern N is given as input. The number of spikes decreases. **G)** Average number of output spikes per trial before learning, in response to pattern P (gray bars) and pattern N (black bars), for 6 experiments with different randomly generated patterns P and N, and different random initial synaptic weights of the neuron. **H)** As in G), for the same experiments, but after learning. The average number of spikes per trial increases after training for pattern P, and decreases for pattern N.

3.3 Methods

We first describe the simple neuron model that we used for the theoretical analysis, and then provide derivations of the equations that were discussed in the preceding section. After that we describe the models for neurons, synapses, and synaptic background activity ("noise") that we used in the computer simulations. Finally we provide technical details to each of the 5 computer simulations that we discussed in the preceding section.

3.3.1 Linear Poisson Neuron Model

In our theoretical analysis, we use a linear Poisson neuron model whose output spike train $S_j^{post}(t)$ is a realization of a Poisson process with the underlying instantaneous firing rate $R_j(t)$. The effect of a spike of presynaptic neuron i at time t' on the membrane potential of neuron j is modeled by an increase in the instantaneous firing rate by an amount $w_{ji}(t')\varepsilon(t-t')$, where ε is a response kernel which models the time course of a postsynaptic potential (PSP) elicited by an input spike. Since STDP according to Izhikevich (2007) has been experimentally confirmed only for excitatory synapses, we will consider plasticity only for excitatory connections and assume that $w_{ji} \geq 0$ for all i and $\varepsilon(s) \geq 0$ for all s. Because the synaptic response is scaled by the synaptic weights, we can assume without loss of generality that the response kernel is normalized to $\int_0^\infty ds\ \varepsilon(s) = 1$. In this linear model, the contributions of all inputs are summed up linearly:

$$R_j(t) = \sum_{i=1}^{n} \int_0^\infty ds\ w_{ji}(t-s)\ \varepsilon(s)\ S_i(t-s)\ , \tag{3.19}$$

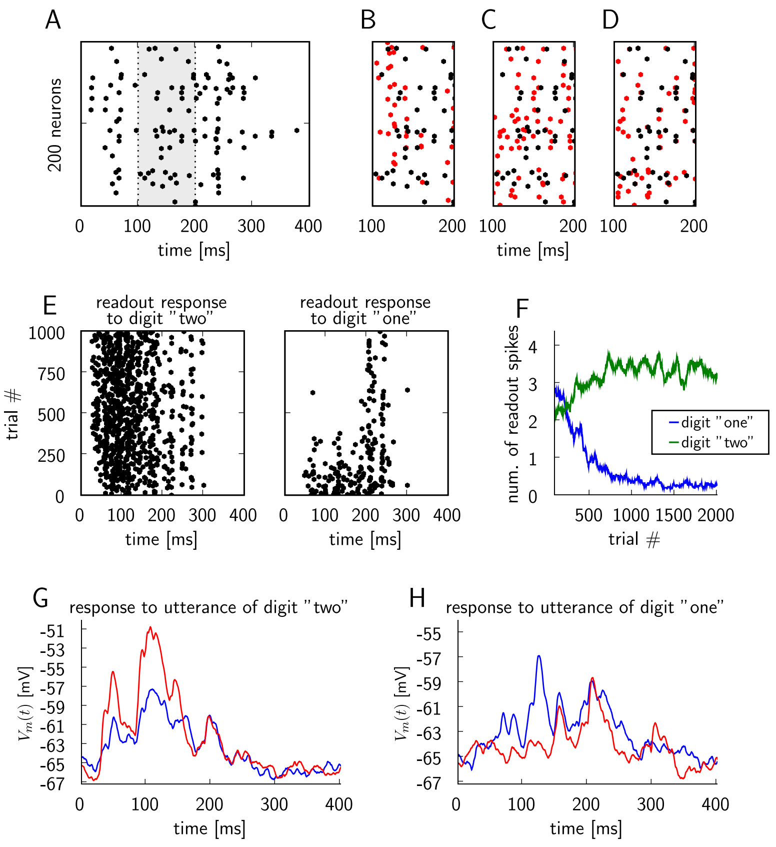

Figure 3.10: See next page for figure caption.

where $S_1, \ldots, S_n$ are the n presynaptic spike trains. Since the instantaneous firing rate $R(t)$ is analogous to the membrane potential of other neuron models, we occasionally refer to $R(t)$ as the "membrane potential" of the neuron.

3.3.2 Learning equations

In the following, we denote by $\langle x \rangle_{E|S_k^{post}(t), S_i^{pre}(t')}$ the ensemble average of a random variable x given that neuron k spikes at time t and neuron i spikes at time t'. We will also sometimes indicate the variables $Y_1, Y_2, \ldots$ over which the average of x is taken by writing $\langle x \rangle_{Y_1, Y_2, \ldots | \ldots}$.

Derivation of equation (3.6). Using equation (3.5), (3.1), and (3.4), we obtain the

Figure 3.10: A LIF neuron is trained through reward-modulated STDP to discriminate as a "readout neuron" responses of generic cortical microcircuits to utterances of different spoken digits. **A)** Circuit response to an utterance of digit "one" (spike trains of 200 out of 540 neurons in the circuit are shown). The response within the time period from 100 to 200 ms (marked in gray) is used as a reference in the subsequent 3 panels. **B)** The circuit response from A) (black) for the period between 100 and 200 ms, and the circuit response to an utterance of digit "two" (red). **C)** The circuit spike response from A) (black) and a circuit response for another utterance of digit "one" (red), also shown for the period between 100 and 200 ms. **D)** The circuit spike response from A) (black), and another circuit response to the same utterance in another trial (red). The responses differ due to the presence of noise in the circuit. **E)** Spike response of the LIF readout neuron for different trials during learning, for trials where utterances of digit "two" (left plot) and digit "one" (right plot) are presented as circuit inputs. The spikes from each 4th trial are plotted. **F)** Average number of spikes in the response of the readout during training, in response to digit "one" (blue) and digit "two" (green). The number of spikes were averaged over 40 trials. **G)** The membrane potential $V_m(t)$ of the neuron during a trial where an input pattern corresponding to an utterance of digit "two" is presented, before (blue curve) and after training (red curve), with the firing threshold removed. **H)** As in G), but for an input pattern corresponding to an utterance of digit "one". The variance of the membrane potential increases during learning for utterances of the rewarded digit, and decreases for the non-rewarded digit.

expected weight change between time t and $t+T$

$$\begin{aligned}
&\frac{\langle w_{ji}(t+T) - w_{ji}(t)\rangle_E}{T} = \\
&\quad \int_0^\infty ds f_c(s) \int_0^\infty dr W(r) \left\langle \left\langle d(t) S_j^{post}(t-s) S_i^{pre}(t-s-r) \right\rangle_T \right\rangle_E + \\
&\quad \int_0^\infty ds\ f_c(s) \int_{-\infty}^0 dr\ W(r) \left\langle \left\langle d(t) S_j^{post}(t-s+r) S_i^{pre}(t-s) \right\rangle_T \right\rangle_E \\
&\quad = \int_0^\infty dr\ W(r) \int_0^\infty ds\ f_c(s) \left\langle \left\langle d(t) S_j^{post}(t-s) S_i^{pre}(t-s-r) \right\rangle_E \right\rangle_T + \\
&\quad \int_{-\infty}^0 dr\ W(r) \int_{|r|}^\infty ds\ f_c(s+r) \left\langle \left\langle d(t) S_j^{post}(t-s) S_i^{pre}(t-s-r) \right\rangle_E \right\rangle_T \\
&\qquad = \int_0^\infty dr\ W(r) \int_0^\infty ds\ f_c(s) \left\langle D_{ji}(t,s,r)\ \nu_{ji}(t-s,r) \right\rangle_T + \\
&\qquad\qquad \int_{-\infty}^0 dr\ W(r) \int_{|r|}^\infty ds\ f_c(s+r) \left\langle D_{ji}(t,s,r)\ \nu_{ji}(t-s,r) \right\rangle_T ,
\end{aligned}$$

with $D_{ji}(t,s,r) = \langle d(t) |$ Neuron j spikes at $t-s$, and neuron i spikes at $t-s-r\rangle_E$, and the joint firing rate $\nu_{ji}(t,r) = \langle S_j(t) S_i(t-r)\rangle_E$ describes correlations between spike timings of neurons j and i. The joint firing rate $\nu_{ji}(t-s,r)$ depends on the weight at time $t-s$. If the learning rate defined by the magnitude of $W(r)$ is small, the synaptic weights can be assumed constant on the time scale of T. Thus, the time scales of neuronal dynamics are separated from the slow time scale of learning. For slow learning, synaptic weights integrate a large number of small changes. We can then expect that averaged quantities enter the learning dynamics. In this case, we can argue that fluctuations of a weight w_{ji} about its mean are negligible and it can well be approximated by its average $\langle w_{ji} \rangle_E$ (it is "self-averaging", see Gerstner and Kistler (2002); Kempter et al. (1999)). To ensure that average quantities enter the learning dynamics, many presynaptic and postsynaptic spikes as well as many independently delivered rewards at varying delays have to occur within T. Hence, in general, the time scale of single spike occurrences and the time scale of the eligibility trace is required to be much smaller than the time scale of learning. If time scales can be separated, we can drop the expectation on the left hand side of the last equation and write

$$\frac{\langle w_{ji}(t+T) - w_{ji}(t) \rangle_E}{T} = \frac{w_{ji}(t+T) - w_{ji}(t)}{T} = \frac{1}{T} \int_t^{t+T} \frac{d}{dt} w_{ji}(t') dt' = \frac{d}{dt} \langle w_{ji}(t) \rangle_T \, .$$

We thus obtain equation (3.6):

$$\begin{aligned} \frac{d}{dt} \langle w_{ji}(t) \rangle_T &= \int_0^\infty dr \; W(r) \int_0^\infty ds \; f_c(s) \, \langle D_{ji}(t,s,r) \; \nu_{ji}(t-s,r) \rangle_T \\ &+ \int_{-\infty}^0 dr \; W(r) \int_{|r|}^\infty ds \; f_c(s+r) \, \langle D_{ji}(t,s,r) \; \nu_{ji}(t-s,r) \rangle_T \, . \end{aligned}$$

Simplification of equation (3.6). In order to simplify this equation, we first observe that $W(r)$ is vanishing for large $|r|$. Hence we can approximate the integral over the learning window by a bounded integral $\int_{-\infty}^{\infty} dr \; W(r) \approx \int_{-T_W}^{T_W} dr \; W(r)$ for some $T_W > 0$ and $T_W \ll T$. In the analyzes of this work, we consider the case where reward is delivered with a relatively large temporal delay. To be more precise, we assume that a pre-post spike pair has an effect on the reward signal only after some minimal delay d_r and that we can write $D_{ji}(t,s,r) = d_0 + D_{ji}^{pre,post}(t,s,r)$ for some baseline reward d_0 and a part which depends on the timing of pre-post spike pairs with $D_{ji}^{pre,post}(t,s,r) = 0$ for $s < d_r$ and $d_r > T_W$. We can then approximate the second term of equation (3.6):

$$\begin{aligned} &\int_{-\infty}^0 dr \; W(r) \int_{|r|}^\infty ds \; f_c(s+r) \, \langle D_{ji}(t,s,r) \; \nu_{ji}(t-s,r) \rangle_T \\ &\quad \approx \int_{-T_W}^0 dr \; W(r) \int_{|r|}^\infty ds \; f_c(s+r) \left\langle (d_0 + D_{ji}^{pre,post}(t,s,r)) \; \nu_{ji}(t-s,r) \right\rangle_T \\ &\qquad\quad \approx \int_{-T_W}^0 dr \; W(r) \left[\int_0^\infty ds \; f_c(s) d_0 \, \langle \nu_{ji}(t-s,r) \rangle_T \right. \\ &\qquad\qquad\qquad \left. + \int_{|r|}^\infty ds \; f_c(s+r) \left\langle D_{ji}^{pre,post}(t,s,r) \; \nu_{ji}(t-s,r) \right\rangle_T \right] \end{aligned}$$

because $\langle \nu_{ji}(t-s-r,r) \rangle_T \approx \langle \nu_{ji}(t-s,r) \rangle_T$ for $r \in [-T_W, T_W]$ and $T_W \ll T$. Since $D_{ji}^{pre,post}(t,s,r) = 0$ for $s \leq T_W$, the second term in the brackets is equivalent to

$\int_0^\infty ds\ f_c(s+r) \left\langle D_{ji}^{pre,post}(t,s,r)\ \nu_{ji}(t-s,r) \right\rangle_T$ which in turn is approximately given by $\int_0^\infty ds\ f_c(s) \left\langle D_{ji}^{pre,post}(t,s,r)\ \nu_{ji}(t-s,r) \right\rangle_T$ if we assume that $f_c(s+r) \approx f_c(s)$ for $s \geq d_r$ and $|r| < T_W$. We can thus approximate the second term of equation (3.6) as

$$\begin{aligned}
&\int_{-\infty}^0 dr\ W(r) \int_{|r|}^\infty ds\ f_c(s+r) \left\langle D_{ji}(t,s,r)\ \nu_{ji}(t-s,r) \right\rangle_T \\
&\qquad\approx \int_{-T_W}^0 dr\ W(r) \left[\int_0^\infty ds\ f_c(s) d_0 \left\langle \nu_{ji}(t-s,r) \right\rangle_T \right. \\
&\qquad\left. + \int_0^\infty ds\ f_c(s) \left\langle D_{ji}^{pre,post}(t,s,r)\ \nu_{ji}(t-s,r) \right\rangle_T \right] \\
&\qquad\approx \int_{-\infty}^0 dr\ W(r) \int_0^\infty ds\ f_c(s) \left\langle D_{ji}(t,s,r)\ \nu_{ji}(t-s,r) \right\rangle_T .
\end{aligned}$$

With this approximation, the first and second term of equation (3.6) can be combined in a single integral to obtain equation (3.8).

3.3.3 Derivations for the biofeedback experiment

We assume that a reward with the functional form ε_r is delivered for each postsynaptic spike with a delay d_r. The reward as time t is therefore

$$d(t) = \int_0^\infty dr\ S_k^{post}(t - d_r - r)\varepsilon_r(r).$$

Weight change for the reinforced neuron (derivation of equation (3.10)). The reward correlation for a synapse ki afferent to the reinforced neuron is

$$\begin{aligned}
D_{ki}(t,s,r) &= \langle d(t) \rangle_{E|S_k^{post}(t-s),S_i^{pre}(t-s-r)} \\
&= \int_0^\infty dr'\ \varepsilon_r(r') \langle S_k^{post}(t-d_r-r') \rangle_{E|S_k^{post}(t-s),S_i^{pre}(t-s-r)} \\
&= \int_0^\infty dr'\ \varepsilon_r(r') \left[\nu_k(t-d_r-r') + w_{ki}\varepsilon(s+r-d_r-r') \right. \\
&\qquad \left. + \delta(s-d_r-r') \right] \\
= \int_0^\infty & dr' \varepsilon_r(r') \nu_k(t-d_r-r') + w_{ki} \int_0^\infty dr'\ \varepsilon_r(r')\varepsilon(s+r-d_r-r') + \varepsilon_r(s-d_r).
\end{aligned}$$

If we assume that the output firing rate is constant on the time scale of the reward function, the first term vanishes. We rewrite the result as

$$D_{ki}(t,s,r) = \varepsilon_r(s-d_r) + w_{ki} \int_{-\infty}^\infty dr'\ \varepsilon_r(s-d_r+r')\varepsilon(r-r').$$

The mean weight change for weights to the reinforced neuron is therefore

$$\frac{d}{dt}w_{ki}(t) = \int_{-\infty}^{\infty} dr\ W(r) \Bigg(\int_{0}^{\infty} ds\ f_c(s)\varepsilon_r(s-d_r)\left\langle \nu_{ki}(t-s,r)\right\rangle_T + \\ w_{ki}\int_{-\infty}^{\infty} dr'\ \varepsilon(r-r') \int_{0}^{\infty} ds\ f_c(s)\varepsilon_r(s-d_r+r')\left\langle \nu_{ki}(t-s,r)\right\rangle_T \Bigg). \quad (3.20)$$

We show that the second term in the brackets is very small compared to the first term:

$$w_{ki}\int_{-\infty}^{\infty} dr'\ \varepsilon(r-r') \int_{0}^{\infty} ds\ f_c(s)\varepsilon_r(s-d_r+r')\left\langle \nu_{ki}(t-s,r)\right\rangle_T = \\ w_{ki}\int_{-\infty}^{\infty} dr'\ \varepsilon(r-r') \int_{0}^{\infty} ds\ f_c(s-r')\varepsilon_r(s-d_r)\left\langle \nu_{ki}(t-s-r',r)\right\rangle_T \approx \\ w_{ki}\int_{-\infty}^{\infty} dr'\ \varepsilon(r-r') \int_{0}^{\infty} ds\ f_c(s)\varepsilon_r(s-d_r)\left\langle \nu_{ki}(t-s,r)\right\rangle_T .$$

The last approximation is based on the assumption that $f_c(s) \approx f_c(s-r')$ and $\langle \nu_{ki}(t-r',r)\rangle_T \approx \langle \nu_{ki}(t,r)\rangle_T$ for $r' \in [-T_W - T_\varepsilon, T_W]$. Here, T_W is the time scale of the learning window (see above), and T_ε is time scale of the PSP, i.e., we have $\varepsilon(s) \approx 0$ for $s \geq T_\varepsilon$. Since $\int_{-\infty}^{\infty} dr\ \varepsilon(r) = 1$ by definition, we see that this is the first term in the brackets of equation (3.20) scaled by w_{ki}. For neurons with many input synapses we have $w_{ki} \ll 1$. Thus the second term in the brackets of equation (3.20) is small compared to the first term. We therefore have

$$\frac{d}{dt}w_{ki}(t) \approx \int_{0}^{\infty} ds\ f_c(s+d_r)\varepsilon_r(s) \int_{-\infty}^{\infty} dr\ W(r)\left\langle \nu_{ki}(t-d_r-s,r)\right\rangle_T .$$

Weight change for non-reinforced neurons (derivation of equation (3.11)). The reward correlation of a synapse ji to a non-reinforced neuron j is given by

$$D_{ji}(t,s,r) = \langle d(t)\rangle_{E|S_j^{post}(t-s),S_i^{pre}(t-s-r)} \\ = \int_{0}^{\infty} dr'\ \varepsilon_r(r')\langle S_k^{post}(t-d_r-r')\rangle_{E|S_j^{post}(t-s),S_i^{pre}(t-s-r)}.$$

We have

$$\langle S_k^{post}(t-d_r-r')\rangle_{E|S_j^{post}(t-s),S_i^{pre}(t-s-r)} \\ = \frac{\langle S_k^{post}(t-d_r-r')S_j^{post}(t-s)\rangle_{E|S_i^{pre}(t-s-r)}}{\langle S_j^{post}(t-s)\rangle_{E|S_i^{pre}(t-s-r)}} \\ = \frac{\nu_{kj}(t-d_r-r', s-d_r-r') + w_{ki}w_{ji}\varepsilon(s+r-d_r-r')\varepsilon(r)}{\nu_j(t-s)+w_{ji}\varepsilon(r)},$$

for which we obtain

$$D_{ji}(t,s,r) = \int_{0}^{\infty} dr'\ \varepsilon_r(r')\frac{\nu_{kj}(t-d_r-r', s-d_r-r') + w_{ki}w_{ji}\varepsilon(s+r-d_r-r')\varepsilon(r)}{\nu_j(t-s)+w_{ji}\varepsilon(r)}.$$

In analogy to the previous derivation, we assume here that the firing rate $\nu_j(t-s)$ in the denominator results from many PSPs. Hence, the single PSP $w_{ji}\varepsilon(r)$ is small compared to $\nu_j(t-s)$. Similarly, we assume that with weights $w_{ki}, w_{ji} \ll 1$, the second term in the nominator is small compared to the joint firing rate $\nu_{kj}(t-d_r-r', s-d_r-r')$. We therefore approximate the reward correlation by

$$D_{ji}(t,s,r) \approx \int_0^\infty dr' \; \varepsilon_r(r') \frac{\nu_{kj}(t-d_r-r', s-d_r-r')}{\nu_j(t-s)}.$$

Hence, the reward correlation of a non-reinforced neuron depends on the correlation of this neuron with the reinforced neuron. The mean weight change for a non-reinforced neuron $j \neq k$ is therefore

$$\frac{d}{dt}w_{ji}(t) \approx \int_0^\infty ds\, f_c(s) \int_{-\infty}^\infty dr\, W(r) \int_0^\infty dr' \varepsilon_r(r') \left\langle \frac{\nu_{kj}(t-d_r-r', s-d_r-r')}{\nu_j(t-s)} \nu_{ji}(t-s,r) \right\rangle_T$$

This equation deserves a remark for the case that $\nu_j(t-s)$ is zero, since it appears in the denominator of the fraction. Note that in this case, both $\nu_{kj}(t-d_r-r', s-d_r-r')$ and $\nu_{ji}(t-s,r)$ are zero. In fact, if we take the limit $\nu_j(t-s) \to 0$, then both of these factors approach zero at least as fast. Hence, in the limit of $\nu_j(t-s) \to 0$, the term in the angular brackets evaluates to zero. This reflects the fact that since STDP is driven by pre- and postsynaptic spikes, there is no weight change if no postsynaptic spikes occur.
For uncorrelated neurons, equation 3.11 evaluates to zero. For uncorrelated neurons k, j, $\nu_{kj}(t-d_r-r', s-d_r-r')$ can be factorized into $\nu_k(t-d_r-r')\nu_j(t-s)$, and we obtain

$$\frac{d}{dt}w_{ji}(t) \approx \int_0^\infty ds\; f_c(s) \int_{-\infty}^\infty dr\; W(r) \int_0^\infty dr' \varepsilon_r(r') \left\langle \nu_k(t-d_r-r')\nu_{ji}(t-s,r) \right\rangle_T.$$

This evaluates approximately to zero if the mean output rate of neuron k is constant on the time scale of the reward kernel.

3.3.4 Analysis of spike-timing dependent rewards (derivation of the conditions (3.13)-(3.15)).

Below, we will indicate the variables $Y_1, Y_2, \ldots$ over which the average of x is taken by writing $\langle x \rangle_{Y_1,Y_2,\ldots|\ldots}$. From equation (3.12), we can determine the reward correlation for synapse i

$$\begin{aligned} D_{ji}(t,s,r) &= \int_{-\infty}^\infty dr' \kappa(r') \left\langle S_j^{post}(t-d_r) S^*(t-d_r-r') \right\rangle_{E|S_j^{post}(t-s), S_i^{pre}(t-s-r)} \\ &= \int_{-\infty}^\infty dr' \kappa(r') \left[\nu_j^{post}(t-d_r) + \delta(s-d_r) + w_{ji}(s+r-d_r)\varepsilon(s+r-d_r)\right] \\ &\qquad \left[\nu^*(t-d_r-r') + w_i^* \varepsilon(s+r-d_r-r')\right], \quad (3.21) \end{aligned}$$

where $\nu_j^{post}(t) = \langle S_j^{post}(t) \rangle_E$ denotes the instantaneous firing rate of the trained neuron at time t, and $\nu^*(t) = \langle S^*(t) \rangle_E$ denotes the instantaneous rate of the target spike train at time t. Since weights are changing very slowly, we have $w_{ji}(t-s-r) \approx w_{ji}(t)$. In the following, we will drop the dependence of w_{ji} on t for brevity. For simplicity, we assume that input rates are stationary and uncorrelated. In this case (since the weights are changing slowly), also the correlations between inputs and outputs can be assumed stationary, $\nu_{ji}(t,r) = \nu_{ji}(r)$. With constant input rates, we can rewrite (3.21) as

$$\begin{aligned} D_{ji}(t,s,r) = \bar{\kappa}\nu^* \nu_j^{post} + \bar{\kappa}\nu^*\delta(s-d_r) + \bar{\kappa}\nu^* w_{ji}\varepsilon(s+r-d_r) \\ + w_i^* \int_{-\infty}^{\infty} dr' \kappa(r')\varepsilon(s+r-d_r-r') \\ \left[\nu_j^{post}(t-d_r) + \delta(s-d_r) + w_{ji}(s+r-d_r)\varepsilon(s+r-d_r)\right], \end{aligned}$$

with $\bar{\kappa} = \int_{-\infty}^{\infty} ds\ \kappa(s)$. We use this results to obtain the temporally smoothed weight change for synapse ji. With stationary correlations, we can drop the dependence of ν_{ji} on t and write $\nu_{ji}(t,r) = \nu_{ji}(r)$. Furthermore, we define $\nu_{ji}^W(r) = \nu_{ji}(r)W(r)$ and obtain

$$\begin{aligned} \frac{d}{dt} w_{ji}(t) &= \int_{-\infty}^{\infty} dr\ W(r)\nu_{ji}(r) \int_0^{\infty} ds\ f_c(s) \left\langle D_{ji}(t,s,r)\right\rangle_T \\ &= \int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r)\bar{\kappa} \left[\nu^*\nu_j^{post}\bar{f}_c + \nu^* f_c(d_r) \right. \\ &\quad \left. +\nu^* w_{ji} \int_0^{\infty} ds\ f_c(s)\varepsilon(s+r-d_r)\right] + \\ &\quad \int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r) w_i^* \nu^{post} \int_{-\infty}^{\infty} dr' \kappa(r') \int_0^{\infty} ds\ f_c(s)\varepsilon(s+r-d_r-r') + \\ &\quad \int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r) w_i^* \int_{-\infty}^{\infty} dr' \kappa(r') f_c(d_r)\varepsilon(r-r') + \\ &\quad \int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r) w_i^* \int_{-\infty}^{\infty} dr' \kappa(r') w_{ji} \int_0^{\infty} ds\ f_c(s)\varepsilon(s+r-d_r)\varepsilon(s+r-d_r-r'). \end{aligned}$$

We assume that the eligibility function $f_c(d_r) \approx f_c(d_r + r)$ if $|r|$ is on the time scale of a PSP, the learning window, or the reward kernel, and that d_r is large compared to these time scales. Then, we have

$$\int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r) \int_{-\infty}^{\infty} dr'\ \kappa(r') f_c(d_r)\varepsilon(r-r') = f_c(d_r) \int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r)\varepsilon_\kappa(r)$$

where $\varepsilon_\kappa(r) = \int_{-\infty}^{\infty} dr'\ \kappa(r')\varepsilon(r-r')$ is the convolution of the reward kernel with the PSP. Furthermore, we find

$$\begin{aligned} &\int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r) \int_{-\infty}^{\infty} dr'\ \kappa(r') \int_0^{\infty} ds\ f_c(s)\varepsilon(s+r-d_r)\varepsilon(s+r-d_r-r') \\ &\approx f_c(d_r) \int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r) \int_{-\infty}^{\infty} dr'\ \kappa(r') \int_0^{\infty} ds\ \varepsilon(s+r-d_r)\varepsilon(s+r-d_r-r') \\ &\qquad = f_c(d_r) \int_{-\infty}^{\infty} dr\ \nu_{ji}^W(r) \int_0^{\infty} ds\ \varepsilon(s)\varepsilon_\kappa(s). \end{aligned}$$

With these simplifications, and the abbreviation $\bar{\nu}_{ji}^{W} = \int_{-\infty}^{\infty} dr \nu_{ji}^{W}(r)$ we obtain the weight change at synapse ji

$$\begin{aligned}\frac{d}{dt} w_{ji}(t) \approx \bar{\kappa} \nu^{*} \nu_{j}^{post} \bar{\nu}_{ji}^{W} \bar{f}_{c} + f_{c}(d_{r}) \bar{\kappa} \bar{\nu}_{ji}^{W} \left[\nu^{*} + \nu^{*} w_{ji} + w_{i}^{*} \nu_{j}^{post}\right] \\ + f_{c}(d_{r}) w_{i}^{*} \int_{-\infty}^{\infty} dr W(r) \nu_{ji}(r) \varepsilon_{\kappa}(r) + f_{c}(d_{r}) w_{ji} w_{i}^{*} \bar{\nu}_{ji}^{W} \int_{-\infty}^{\infty} dr\, \varepsilon(r) \varepsilon_{\kappa}(r),\end{aligned}$$

where $\bar{\nu}_{ji}^{W} = \int_{-\infty}^{\infty} dr W(r) \nu_{ji}(r)$.

For uncorrelated Poisson input spike trains of rate ν_{i}^{pre} and the linear Poisson neuron model, the input-output correlations are $\nu_{ji}(r) = \nu_{i}^{pre} \nu_{j}^{post} + w_{ji} \nu_{i}^{pre} \varepsilon(r)$. With these correlations, we obtain $\bar{\nu}_{ji}^{W} = \nu_{i}^{pre} \nu_{j}^{post} \bar{W} + w_{ji} \nu_{i}^{pre} \bar{W}_{\varepsilon}$ where $\bar{W} = \int_{-\infty}^{\infty} dr\, W(r)$, and $\bar{W}_{\varepsilon} = \int_{-\infty}^{\infty} dr\, \varepsilon(r) W(r)$. The weight change at synapse ji is then

$$\begin{aligned}\frac{d}{dt} w_{ji}(t) \quad \approx \quad & \bar{\kappa} \bar{f}_{c} \nu^{*} \nu_{i}^{pre} \nu_{j}^{post} \left[\nu_{j}^{post} \bar{W} + w_{ji} \bar{W}_{\varepsilon}\right] \\ & + \bar{\kappa} f_{c}(d_{r}) \nu_{i}^{pre} \left[\nu_{j}^{post} \bar{W} + w_{ji} \bar{W}_{\varepsilon}\right] \left[\nu^{*} + \nu^{*} w_{ji} + w_{i}^{*} \nu_{j}^{post}\right] \\ & + f_{c}(d_{r}) w_{i}^{*} \nu_{i}^{pre} \left[\nu_{j}^{post} \int_{-\infty}^{\infty} dr\, W(r) \varepsilon_{\kappa}(r) + w_{ji} \int_{-\infty}^{\infty} dr\, W(r) \varepsilon(r) \varepsilon_{\kappa}(r)\right] \\ & + f_{c}(d_{r}) w_{i}^{*} w_{ji} \nu_{i}^{pre} \left[\nu_{j}^{post} \bar{W} + w_{ji} \bar{W}_{\varepsilon}\right] \int_{0}^{\infty} dr\, \varepsilon(r) \varepsilon_{\kappa}(r), \qquad (3.22)\end{aligned}$$

We will now bound the expected weight change for synapses ji with $w_{i}^{*} = w_{max}$ and for synapses jk with $w_{k}^{*} = 0$. In this way we can derive conditions for which the expected weight change for the former synapses is positive, and that for the latter type is negative. First, we assume that the integral over the reward kernel is positive. In this case, the weight change given by (3.22) is negative for synapses i with $w_{i}^{*} = 0$ if and only if $\nu_{i}^{pre} > 0$, and $-\nu_{j}^{post} \bar{W} > w_{ji} \bar{W}_{\varepsilon}$. In the worst case, w_{ji} is w_{max} and ν_{j}^{post} is small. We have to guarantee some minimal output rate ν_{min}^{post} such that even if $w_{ji} = w_{max}$, this inequality is fulfilled. This could be guaranteed by some noise current. Given such minimal output rate, we can state the first inequality which guarantees convergence of weights w_{ji} with $w_{i}^{*} = 0$

$$-\nu_{min}^{post} \bar{W} > w_{max} \bar{W}_{\varepsilon}.$$

For synapses ji with $w_{i}^{*} = w_{max}$, we obtain two more conditions. The approximate weight change is given by

$$\begin{aligned}\frac{d}{dt}w_{ji}(t)\frac{1}{\nu_i^{pre}} \approx\ & \bar{\kappa}\left[\nu_j^{post}\bar{W} + w_{ji}\bar{W}_\varepsilon\right] \\ & \left[\nu^*\nu_j^{post}\bar{f}_c + f_c(d_r)\nu^* + f_c(d_r)\nu^* w_{ji} + f_c(d_r)\nu_j^{post} w_{max}\right] \\ & + f_c(d_r) w_{max}\nu_j^{post}\int_{-\infty}^{\infty} dr\ W(r)\varepsilon_\kappa(r) \\ & + f_c(d_r) w_{max} w_{ji}\int_{-\infty}^{\infty} dr\ W(r)\varepsilon(r)\varepsilon_\kappa(r) \\ & + f_c(d_r) w_{max} w_{ji}\nu_j^{post}\bar{W}\int_0^{\infty} dr\ \varepsilon(r)\varepsilon_\kappa(r) \\ & + f_c(d_r) w_{max} w_{ji}^2\bar{W}_\varepsilon\int_0^{\infty} dr\ \varepsilon(r)\varepsilon_\kappa(r).\end{aligned}$$

The last term in this equation is positive and small. We can ignore it in our sufficient condition. The second to last term is negative. We will include in our condition that the third to last term compensates for this negative term. Hence, the second condition is

$$\int_{-\infty}^{\infty} dr\ W(r)\varepsilon(r)\varepsilon_\kappa(r) \geq -\nu_j^{post}\bar{W}\int_0^{\infty} dr\ \varepsilon(r)\varepsilon_\kappa(r),$$

which should be satisfied in most setups. If we assume that this holds, we obtain

$$\begin{aligned}\frac{d}{dt}w_{ji}(t) \geq\ & \bar{\kappa}\left[\nu_j^{post}\bar{W} + w_{ji}\bar{W}_\varepsilon\right]\left[\nu^*\nu_j^{post}\bar{f}_c + f_c(d_r)\nu^* + f_c(d_r)\nu^* w_{ji} + f_c(d_r)\nu_j^{post} w_{max}\right] \\ & + f_c(d_r) w_{max}\nu_j^{post}\int_{-\infty}^{\infty} dr\ W(r)\varepsilon_\kappa(r).\end{aligned}$$

which should be positive. We obtain the following inequality

$$\int_{-\infty}^{\infty} dr\ W(r)\varepsilon_\kappa(r) > -\bar{W}\bar{\kappa}\left[\frac{\nu^*\nu_j^{post}}{w_{max}}\frac{\bar{f}_c}{f_c(d_r)} + \frac{\nu^*}{w_{max}} + \nu^* + \nu^{post}\right].$$

All three inequalities are summarized in the following:

$$\begin{aligned}-\nu_{min}^{post}\bar{W} &> w_{max}\bar{W}_\varepsilon \\ \int_{-\infty}^{\infty} dr\ W(r)\varepsilon(r)\varepsilon_\kappa(r) &\geq -\nu_{max}^{post}\bar{W}\int_0^{\infty} dr\ \varepsilon(r)\varepsilon_\kappa(r) \\ \int_{-\infty}^{\infty} dr\ W(r)\varepsilon_\kappa(r) &> -\bar{W}\bar{\kappa}\left[\frac{\nu^*\nu_{max}^{post}}{w_{max}}\frac{\bar{f}_c}{f_c(d_r)} + \frac{\nu^*}{w_{max}} + \nu^* + \nu_{max}^{post}\right],\end{aligned}$$

where ν_{max}^{post} is the maximal output rate. If these inequalities are fulfilled and input rates are positive, then the weight vector converges on average from any initial weight vector to $\mathbf{w}^*$. The second condition is less severe, and should be easily fulfilled in most setups. If this is the case, the first condition (3.13) ensures that weights with $w^* = 0$ are depressed while the third condition (3.15) ensures that weights with $w^* = w_{max}$ are potentiated.

3.3.5 Analysis of the pattern discrimination task (derivation of equation (3.17)).

We assume that a trial consists of the presentation of a single pattern starting at time $t = 0$. We compute the weight change for a single trial given that pattern $X \in \{P, N\}$ was presented with the help of equations (3.1), (3.3), and (3.4) as

$$\begin{aligned}
\left.\frac{d}{dt}w_i(t)\right|_X &= \int_0^\infty ds f_c(s) \left[\int_0^\infty dr W(r) S^{post}(t-s)\delta(t-s-r-t_i^X)\right.\\
&+ \left.\int_0^\infty dr W(-r) S^{post}(t-s-r)\delta(t-s-t_i^X)\right] d(t)\\
&= \alpha^X \int_0^\infty ds f_c(s) \left[\int_0^\infty dr W(r) S^{post}(t-s)\delta(t-s-r-t_i^X)\right.\\
&+ \left.\int_0^\infty dr W(-r) S^{post}(t-s-r)\delta(t-s-t_i^X)\right] \int_0^\infty dr' \varepsilon_r(r') S^{post}(t-d_r-r')\\
&= \alpha^X \int_0^\infty dr f_c(t-r-t_i^X) W(r) \int_0^\infty dr' \varepsilon_r(r') S^{post}(r+t_i^X) S^{post}(t-d_r-r')\\
&+ \alpha^X \int_0^\infty dr f_c(t-t_i^X) W(-r) \int_0^\infty dr' \varepsilon_r(r') S^{post}(t_i^X - r) S^{post}(t-d_r-r').
\end{aligned}$$

We can compute the average weight change given that pattern X was presented:

$$\begin{aligned}
\left\langle\frac{d}{dt}w_i(t)\right\rangle_{E|X} &= \alpha^X \int_0^\infty dr f_c(t-r-t_i^X)\\
&\quad W(r) \int_0^\infty dr' \varepsilon_r(r') \langle S^{post}(t_i^X + r) S^{post}(t-d_r-r')\rangle_{E|X}\\
&+ \alpha^X \int_0^\infty dr f_c(t-t_i^X)\\
&\quad W(-r) \int_0^\infty dr' \varepsilon_r(r') \langle S^{post}(t_i^X - r) S^{post}(t-d_r-r')\rangle_{E|X}.
\end{aligned}$$

If we assume that f_c is approximately constant on the time scale of the learning window W, we can simplify this to

$$\left\langle\frac{d}{dt}w_i(t)\right\rangle_{E|X} = \int_{-\infty}^\infty dr f_c(t-t_i^X) W(r) \int_0^\infty dr' \varepsilon_r(r') \langle S^{post}(t_i^X + r) S^{post}(t-d_r-r')\rangle_{E|X} \alpha^X.$$

For the linear Poisson neuron, we can write the auto-correlation function as

$$\begin{aligned}
\langle S^{post}(t_i^X + r) S^{post}(t-d_r-r')\rangle_{E|X} &= [\nu^X(t_i^X + r)\nu^X(t-d_r-r')\\
&\quad + \nu^X(t_i^X + r)\delta(t_i^X + r - t + d_r + r')]\\
&= \nu^X(t_i^X + r)[\nu^X(t-d_r-r') +\\
&\quad \delta(t_i^X + r - t + d_r + r')],
\end{aligned}$$

where $\nu^X(t) = \langle S^{post}(t)\rangle_{E|X}$ is the ensemble average rate at time t given that pattern X was presented. If an experiment for a single pattern runs over the time interval

$[0, T']$, we can compute the total average weight change Δw_i^X of a trial given that pattern X was presented as

$$\begin{aligned}
\Delta w_i^X &= \int_0^{T'} dt \left\langle \frac{d}{dt} w_i(t) \right\rangle_{E|X} \\
&= \alpha^X \int_{-\infty}^{\infty} dr W(r) \nu^X(t_i^X + r) \int_0^{T'} dt f_c(t - t_i^X) \int_0^{\infty} dr' \varepsilon_r(r') \\
&\quad [\nu^X(t - d_r - r') + \delta(t_i^X + r - t + d_r + r')] \\
&= \alpha^X \int_{-\infty}^{\infty} dr W(r) \nu^X(t_i^X + r) \int_0^{\infty} dr' \varepsilon_r(r') \\
&\quad \left[f_c(r + d_r + r') + \int_{d_r}^{T'} dt f_c(t - t_i^X) \nu^X(t - d_r - r') \right] \\
&\approx \alpha^X \int_{-\infty}^{\infty} dr W(r) \nu^X(t_i^X + r) \int_0^{\infty} dr' \varepsilon_r(r') \\
&\quad \left[f_c(d_r + r') + \int_0^{T'} dt f_c(t - t_i^X) \nu^X(t - d_r - r') \right] \qquad (3.23)
\end{aligned}$$

By defining

$$A_i^X = \alpha^X \int_0^{\infty} dr' \varepsilon_r(r') \left[f_c(d_r + r') + \int_0^{T'} dt f_c(t - t_i^X) \nu^X(t - d_r - r') \right],$$

we can write equation (3.23) as

$$\Delta w_i^X = \int_{-\infty}^{\infty} dr W(r) \nu^X(t_i^X + r) A_i^X .$$

We assume that eligibility traces and reward signals have settled to zero before a new pattern is presented. The expected weight change for the successive presentation of both patterns is therefore

$$\Delta w_i = \int_{-\infty}^{\infty} dr W(r) \left[\nu^P(t_i^P + r) A_i^P + \nu^N(t_i^N + r) A_i^N \right].$$

The equations can easily be generalized to the case where multiple input spikes per synapse are allowed and where jitter on the templates is allowed. However, the main effect of the rule can be read off the equations given here.

3.3.6 Common models and parameters of the computer simulations

We describe here the models and parameter values that were used in all our computer simulations. We will specify in a subsequent section the values of other parameters that had to be chosen differently in individual computer simulations, in dependence of their different setups and requirements of each computer simulation.

3.3.6.1 LIF neuron model

For the computer simulations LIF neurons with conductance-based synapses were used. The membrane potential $V_m(t)$ of this neuron model is given by:

$$C_m \frac{dV_m(t)}{dt} = -\frac{V_m(t) - V_{resting}}{R_m} - \sum_{j=1}^{K_e} g_{e,j}(t)(V_m(t) - E_e) - \sum_{j=1}^{K_i} g_{i,j}(V_m(t) - E_i) - I_{noise}(t), \tag{3.24}$$

where C_m is the membrane capacitance, R_m is the membrane resistance, $V_{resting}$ is the resting potential, and $g_{e,j}(t)$ and $g_{i,j}(t)$ are the K_e and K_i synaptic conductances from the excitatory and inhibitory synapses respectively. The constants E_e and E_i are the reversal potentials of excitatory and inhibitory synapses. I_{noise} represents the synaptic background current which the neuron receives (see below for details).

Whenever the membrane potential reaches a threshold value V_{thresh}, the neuron produces a spike, and its membrane potential is reset to the value of the reset potential V_{reset}. After a spike, there is a refractory period of length $T_{refract}$, during which the membrane potential of the neuron remains equal to the value $V_m(t) = V_{reset}$. After the refractory period $V_m(t)$ continues to change according to equation (3.24).

For a given synapse, the dynamics of the synaptic conductance $g(t)$ is defined by

$$\frac{dg(t)}{dt} = -\frac{g(t)}{\tau_{syn}} + \sum_k A(t^{(k)} + t_{delay})\delta(t - t^{(k)} - t_{delay}) \quad , \tag{3.25}$$

where $A(t)$ is the amplitude of the postsynaptic response (PSR) to a single presynaptic spike, which varies over time due to the inherent short-term dynamics of the synapse, and $\{t^{(k)}\}$ are the spike times of the presynaptic neuron. The conductance of the synapse decreases exponentially with time constant τ_{syn}, and increases instantaneously by amount of $A(t)$ whenever the presynaptic neuron spikes.

In all computer simulations we used the following values for the neuron and synapse parameters. The membrane resistance of the neurons was $R_m = 100M\Omega$, the membrane capacitance $C_m = 0.3nF$, the resting potential, reset potential and the initial value of the membrane potential had the same value of $V_{resting} = V_{reset} = V_m(0) = -70$mV, the threshold potential was set to $V_{thresh} = -59mV$ and the refractory period $T_{refract} = 5$ms. For the synapses we used a time constant set to $\tau_{syn} = 5$ms, reversal potential $E_e = 0$ mV for the excitatory synapses and $E_e = -75$ mV for the inhibitory synapses. All synapses had a synaptic delay of $t_{delay} = 1$ms.

3.3.6.2 Short-term dynamics of synapses

We modeled the short-term dynamics of synapses according to the phenomenological model proposed in Markram et al. (1998), where the amplitude $A_k = A(t_k + t_{delay})$ of the postsynaptic response for the k^{th} spike in a spike train with inter-spike intervals

source/dest.	exc.(U,D,F)	inh. (U,D,F)
exc.	0.5, 1.1, 0.02	0.25, 0.7, 0.02
inh.	0.05, 0.125, 1.2	0.32, 0.144, 0.06

Table 3.2: Mean values of the U, D and F parameters in the model from Markram et al. (1998) for the short-term dynamics of synapses, depending on the type of the presynaptic and postsynaptic neuron (excitatory or inhibitory). These mean values, based on experimental data from Markram et al. (1998); Gupta et al. (2000), were used in all computer simulations.

$\Delta_1, \Delta_2, \ldots, \Delta_{k-1}$ is calculated with the following equations

$$\begin{aligned} A_k &= w \cdot u_k \cdot R_k \\ u_k &= U + u_{k-1}(1-U)e^{-\Delta_{k-1}/F} \\ R_k &= 1 + (R_{k-1} - u_{k-1}R_{k-1} - 1)e^{-\Delta_{k-1}/D}, \end{aligned} \tag{3.26}$$

with hidden dynamic variables $u \in [0,1]$ and $R \in [0,1]$ whose initial values for the 1^{st} spike are $u_1 = U$ and $R = 1$ (see Maass and Markram (2002) for a justification of this version of the equations, which corrects a small error in Markram et al. (1998)). The variable w is the synaptic weight which scales the amplitudes of postsynaptic responses. If long-term plasticity is introduced, this variable is a function of time. In the simulations, for the neurons in the circuits the values for the U, D and F parameters were drawn from Gaussian distributions with mean values which depended on whether the type of presynaptic and postsynaptic neuron of the synapse is excitatory or inhibitory, and were chosen according to the data reported in Markram et al. (1998) and Gupta et al. (2000). The mean values of the Gaussian distributions are given in Table 3.2, and the standard deviation was chosen to be 50% of its mean. Negative values were replaced with values drawn from uniform distribution with a range between 0 and twice the mean value. For the simulations involving individual trained neurons, the U, D and F parameters of these neurons were set to the values from Table 3.2.

We have carried out control experiments with current-based synapses that were not subject to short-term plasticity (see Fig. S5, S8, S9; successful control experiments with static current-based synapses were also carried out for computer simulation 1, results not shown). We found that the results of all our computer simulations also hold for static current-based synapses.

3.3.6.3 Model of background synaptic activity

To reproduce the background synaptic input cortical neurons receive in vivo, the neurons in our models received an additional noise process as conductance input. The noise process we used is a point-conductance approximation model, described in Destexhe et al. (2001). According to Destexhe et al. (2001), this noise process models the effect of a bombardment by a large number of synaptic inputs in vivo, which causes membrane potential depolarization, referred to as “high conductance” state. Furthermore, it was shown that it captures the spectral and amplitude characteristics of the input conductances of a detailed biophysical model of a neocortical pyramidal cell that was matched to intracellular recordings in cat parietal cortex in vivo. The ratio of average contributions of excitatory and inhibitory background conductances was chosen to be 5 in accordance to experimental studies during sensory responses (see Borg-Graham et al. (1998),Hirsch et al. (1998), and Anderson et al. (2000)). In this model, the noisy synaptic current I_{noise} in equation (3.24) is a sum of two currents:

$$I_{noise}(t) = g_e(t)(V_m(t) - E_e) + g_i(t)(V_m(t) - E_i), \tag{3.27}$$

where $g_e(t)$ and $g_i(t)$ are time-dependent excitatory and inhibitory conductances. The values of the respective reversal potentials were $E_e = 0$ mV and $E_i = -75$ mV. The conductances $g_e(t)$ and $g_i(t)$ were modeled according to Destexhe et al. (2001) as a one-variable stochastic process similar to an Ornstein-Uhlenbeck process:

$$\begin{aligned}\frac{dg_e(t)}{dt} &= -\frac{1}{\tau_e}[g_e(t) - g_{e0}] + \sqrt{D_e}\chi_1(t) \\ \frac{dg_i(t)}{dt} &= -\frac{1}{\tau_i}[g_i(t) - g_{i0}] + \sqrt{D_i}\chi_2(t),\end{aligned}$$

with mean $g_{e0} = 0.012\mu$S, noise-diffusion constant $D_e = 0.003\mu$S and time constant $\tau_e = 2.7$ms for the excitatory conductance, and mean $g_{i0} = 0.057\mu$S, noise-diffusion constant $D_i = 0.0066\mu$S, and time constant $\tau_i = 10.5$ms for the inhibitory conductance. $\chi_1(t)$ and $\chi_2(t)$ are Gaussian white noise of zero mean and unit standard deviation.

Since these processes are Gaussian stochastic processes, they can be numerically integrated by an exact update rule:

$$\begin{aligned}g_e(t+\Delta) &= g_{e0} + [g_e(t) - g_{e0}]e^{-\frac{\Delta}{\tau_e}} + A_e N_1(0,1) \\ g_i(t+\Delta) &= g_{i0} + [g_i(t) - g_{i0}]e^{-\frac{\Delta}{\tau_i}} + A_i N_2(0,1),\end{aligned}$$

where $N_1(0,1)$ and $N_2(0,1)$ are normal random numbers (zero mean, unit standard deviation) and A_e, A_i are amplitude coefficients given by:

$$\begin{aligned}A_e &= \sqrt{\frac{D_e\tau_e}{2}[1 - e^{\frac{-2\Delta}{\tau_e}}]} \\ A_i &= \sqrt{\frac{D_i\tau_i}{2}[1 - e^{\frac{-2\Delta}{\tau_i}}]}.\end{aligned}$$

3.3.6.4 Reward-modulated STDP

For the computer simulations we used the following parameters for the STDP window function $W(r)$: $A_+ = 0.01w_{max}$, $A_-/A_+ = 1.05$, $\tau_+ = \tau_- = 30$ms. w_{max} denotes the hard bound of the synaptic weight of the particular plastic synapse. Note that the parameter A_+ can be given arbitrary value in this plasticity rule, since it can be scaled together with the reward signal, i.e. multiplying the reward signal by some constant and dividing A_+ by the same constant results in identical time evolution of the weight changes. We have set A_+ to be 1% of the maximum synaptic weight.

We used the α-function to model the eligibility trace kernel $f_c(t)$

$$f_c(t) = \begin{cases} \frac{t}{\tau_e} e^{-\frac{t}{\tau_e}} & , \quad \text{if } t > 0 \\ 0 & , \quad \text{otherwise} \end{cases} , \tag{3.28}$$

where the time constant τ_e was set to $\tau_e = 0.4$s in all computer simulations.

For computer simulations 1 and 4 we performed control experiments (see Fig. S3, S4 and S7) with the weight-dependent synaptic update rule proposed in Morrison et al. (2007), instead of the purely additive rule (3.3). We used the parameters proposed in Morrison et al. (2007), i.e. $\mu = 0.4$, $\alpha = 0.11$, $\tau_+ = \tau_- = 20$ms. The w_0 parameter was calculated according to the formula: $w_0 = \frac{1}{2} w_{max} \alpha^{1/1-\mu}$ where w_{max} is the maximum synaptic weight of the synapse. $\frac{1}{2} w_{max}$ is equal to the initial synaptic weight for the circuit neurons, or to the mean of the distribution of the initial weights for the trained neurons.

3.3.6.5 Initial weights of trained neurons

The synaptic weights of excitatory synapses to the trained neurons in experiments 2-5 were initialized from a Gaussian distribution with mean $w_{max}/2$. The standard deviation was set to $w_{max}/10$ bounded within the range $[3w_{max}/10, 7w_{max}/10]$.

3.3.6.6 Software

All computer simulations were carried out with the PCSIM software package (http://www.lsm.tugraz.at/pcsim). PCSIM is a parallel simulator for biologically realistic neural networks with a fast c++ simulation core and a Python interface. It has been developed by Thomas Natschläger and Dejan Pecevski. The time step of simulation was set to 0.1ms.

3.3.7 Details to individual computer simulations

For all computer simulations, both for the cortical microcircuits and readout neurons, the same parameters values for the neuron and synapse models and the reward-modulated STDP rule were used, as specified in the previous section (except in computer simulation 3, where the goal was to test the theoretical predictions for different values of the parameters). Each of the computer simulations in this work modeled a specific task or experimental finding. Consequently, the dependence of the reward signal on the behavior of the system had to be modeled in a specific way for each simulation (a more detailed discussion of the reward signal can be found in the Discussion section). The parameters for that are given below in separate subsections which address the individual simulations. Furthermore, some of the remaining parameters in the experiments, i.e. the values of the synaptic weights, the

Cortical microcircuits					
simulation No.	neurons	$p_{ee}, p_{ei}, p_{ei}, p_{ii}$	$w_{exc}(0)$ $[nS]$	$w_{inh}(0)$ $[nS]$	C_{OU}
1	4000	0.02,0.02,0.024,0.016	10.7	211.6	1.0, 0.2
5	540	0.1	0.784	5.1	0.4

Table 3.3: Specific parameter values for the cortical microcircuits in computer simulation 1 and 5. p_{conn} is the connection probability, $w_{exc}(0)$ and $w_{inh}(0)$ are the initial synaptic weights for the excitatory and inhibitory synapses respectively, and C_{OU} is the scaling factor for the Ornstein-Uhlenbeck noise injected in the neurons.

Trained (readout) neurons			
simulation No.	num. synapses	w_{max} $[nS]$	C_{OU}
2	100	11.9	1.0
4	200	5.73	0.2
5	432	2.02	0.2

Table 3.4: Specific parameter values for the trained neurons in computer simulation 2, 4 and 5. w_{max} is the upper hard bound of the synaptic weights of the synapses. C_{OU} is the scaling factor for the Ornstein-Uhlenbeck noise injected in the neurons.

number of synapses of a neuron, number of neurons in the circuit and the Ornstein-Uhlenbeck (OU) noise levels were chosen to achieve different goals depending on the particular experiment. Briefly stated, these values were tuned to achieve a certain level of firing activity in the neurons, a suitable dynamical regime of the activity in the circuits, and a specific ratio between amount of input the neurons receive from the input synapses and the input generated by the noise process.

We carried out two types of simulations: simulations of cortical microcircuits in computer simulations 1 and 5, and training of readout neurons in computer simulations 2, 3, 4 and 5. In the following we discuss these two types of simulations in more detail.

3.3.7.1 Cortical Microcircuits

The values of the initial weights of the excitatory and inhibitory synapses for the cortical microcircuits are given in Table 3.3. All synaptic weights were bounded in the range between 0 and twice the initial synaptic weight of the synapse.

The cortical microcircuit was composed of 4000 neurons connected randomly with connection probabilities described in Details to computer simulation 1. The initial synaptic weights of the synapses and the levels of OU noise were tuned to achieve a spontaneous firing rate of about 4.6 Hz, while maintaining an asynchronous irregular firing activity in the circuit. 50% of all neurons (randomly chosen, 50% excitatory and 50% inhibitory) received downscaled OU noise (by a factor 0.2 from the model reported in Destexhe et al. (2001)), with the subtracted part substituted by additional synaptic input from the circuit. The input connection probabilities of these neurons were scaled up, so that the firing rates remain in the same range as for the other neurons. This was done in order to observe how the learning mechanisms work when most of the input conductance in the neuron comes from a larger number of input synapses which are plastic, rather than from a static noise process. The reinforced neurons were randomly chosen from this group of neurons.

We chose a smaller microcircuit, composed of 540 neurons, for the computer simulation 5 in order to be able to perform a large number of training trials. The synaptic weights in this smaller circuit were chosen (see Table 3.3) to achieve an appropriate level of firing activity in the circuit that is modulated by the external input. The circuit neurons had injected an Ornstein-Uhlenbeck (OU) noise multiplied by 0.4 in order to emulate the background synaptic activity in neocortical neurons in vivo, and test the learning in a more biologically realistic settings. This produced significant trial-to-trial variability in the circuit response (see Fig. 3.10D). A lower value of the noise level could also be used without affecting the learning, whereas increasing the amount of injected noise would slowly deteriorate the information that the circuit activity maintains about the injected inputs, resulting in a decline of the learning performance.

3.3.7.2 Readout neurons

The maximum values of the synaptic weights of readout neurons for computer simulations 2, 4 and 5, together with the number of synapses of the neurons, are given in Table 3.4.

The neuron in computer simulation 2 had 100 synapses. We chose 200 synapses for the neuron in computer simulation 4, in order to improve the learning performance. Such improvement of the learning performance for larger numbers of synapses is in accordance with our theoretical analysis (see equation (3.17), since for learning the classification of temporal patterns the temporal variation of the voltage of the postsynaptic membrane turns out to be of critical importance (see the discussion after equation (3.17)). This temporal variation depends less on the shape of a single EPSP and more on the temporal pattern of presynaptic firing when the number of synapses is increased. In computer simulation 5 the readout neuron received inputs from all 432 excitatory neurons in the circuit. The synaptic weights were chosen in accordance with the number of synapses in order to achieve a firing rate suitable for the particular task, and to balance the synaptic input and the noise injections in the neurons.

For the pattern discrimination task (computer simulation 4) and the speech recognition task (computer simulation 5), the amount of noise had to be chosen to be high enough to achieve sufficient variation of the membrane potential from trial to trial near the firing threshold, and low enough so that it would not dominate the fluctuations of the membrane potential. In the experiment where the exact spike times were rewarded (computer simulation 2), the noise had a different role. As described in the Results section, there the noise effectively controls the amount of depression. If the noise (and therefore the depression) is too weak, $w^* = 0$ synapses do not converge to 0. If the noise is too strong, $w^* = w_{max}$ synapses do not converge to w_{max}. To achieve the desired learning result, the noise level should be in a range where it reduces the correlations of the synapses with $w^* = 0$ so that the depression of STDP will prevail, but at the same time is not strong enough to do the same for the other group of synapses with $w^* = w_{max}$, since they have stronger pre-before-post correlations. For our simulations, we have set the noise level to the full amount of OU noise.

3.3.7.3 Details to computer simulation 1: Model for biofeedback experiment

The cortical microcircuit model consisted of 4000 neurons with twenty percent of the neurons randomly chosen to be inhibitory, and the others excitatory. The connections between the neurons were created randomly, with different connectivity probabilities depending on whether the postsynaptic neuron received the full amount of OU noise, or downscaled OU noise with an additional compensatory synaptic input from the circuit. For neurons in the latter sub-population, the connection probabilities were $p_{ee} = 0.02$, $p_{ei} = 0.02$, $p_{ie} = 0.024$ and $p_{ii} = 0.016$ where the ee, ei, ie, ii indices designate the type of the presynaptic and postsynaptic neurons (e=excitatory or i=inhibitory). For the other neurons the corresponding connection probabilities were downscaled by 0.4. The resulting firing rates and correlations for both types of excitatory neurons are plotted in Fig. S1 and S2.

The shape of the reward kernel $\varepsilon_r(t)$ was chosen as a difference of two α-functions

$$\varepsilon_r(t) = A_r^+ \frac{t}{\tau_r^+} e^{1-\frac{t}{\tau_r^+}} - A_r^- \frac{t}{\tau_r^-} e^{1-\frac{t}{\tau_r^-}}, \tag{3.29}$$

one positive α-pulse with a peak at 0.4 sec after the corresponding spike, and one long-tailed negative α-pulse which makes sure that the integral over the reward kernel is zero. The parameters for the reward kernel were $A_r^+ = 1.379$, $A_r^- = 0.27$, $\tau_r^+ = 0.2$s, $\tau_r^- = 1$s, and $d_r = 0.2$s, which produced a peak value of the reward pulse 0.4s after the spike that caused it.

3.3.7.4 Details to computer simulation 2: Learning spike times

We used the following function for the reward kernel $\kappa(r)$

$$\kappa(r) = \begin{cases} A_+^{\kappa}(e^{-\frac{t-t_\kappa}{\tau_1^\kappa}} - e^{-\frac{t-t_\kappa}{\tau_2^\kappa}}) & , \quad \text{if } t - t_\kappa \geq 0 \\ -A_-^{\kappa}(e^{\frac{t-t_\kappa}{\tau_1^\kappa}} - e^{\frac{t-t_\kappa}{\tau_2^\kappa}}) & , \quad \text{otherwise} \end{cases} \tag{3.30}$$

where A_+^κ and A_-^κ are positive scaling constants, τ_1^κ and τ_2^κ define the shape of the two double-exponential functions the kernel is composed of, and t_κ defines the offset of the zero-crossing from the origin. The parameter values used in our simulations were $A_+^\kappa = 0.1457$, $A_-^\kappa = -0.1442$, $\tau_1^\kappa = 30$ms, $\tau_2^\kappa = 4$ms and $t_\kappa = -1$ms. The reward delay was equal to $d_r = 0.4$s.

3.3.7.5 Details to computer simulation 3: Testing the analytically derived conditions

We used a linear Poisson neuron model as in the theoretical analysis with static synapses and exponentially decaying postsynaptic responses $\varepsilon(s) = e^{(-s/\tau_\varepsilon)}/\tau_\varepsilon$. The neuron had 100 excitatory synapses, except in experiment #6, where we used 200 synapses. In all experiments the target neuron received additional 10 excitatory synapses with weights set to w_{max}. The input spike trains were Poisson processes with a constant rate of $r_{pre} = 6$Hz, except in experiment # 6 where the rate was $r_{pre} = 3$Hz. The weights of the target neuron were set to $w_i^* = w_{max}$ for $0 \leq i < 50$ and $w_i^* = 0$ for $50 \leq i < 100$.

The time constants of the reward kernel were $\tau_2^\kappa = 4$ms, whereas τ_1^κ had different values in different experiments (reported in table 3.1). The value of t_κ was always set to an optimal value such that the $\varepsilon_\kappa(0) = \int_0^\infty \kappa(-s)\varepsilon(s) = 0$. The time constant τ_- of the negative part of the STDP window function $W(r)$ was set to τ_+. The reward signal was delayed by $\tau_d = 0.4$s. The simulations were performed for varying durations of simulated biological time (see the t_{sim}-column in Table 3.1).

3.3.7.6 Details to computer simulation 4: Learning pattern classification

We used the reward signal from equation (3.16), with an α-function for the reward kernel $\varepsilon_r(r) = \frac{e}{\tau} t e^{-t/\tau}$, and the reward delay d_r set to 300 ms. The amplitudes of the positive and negative pulses were $\alpha_P = -\alpha_N = 1.435$. and the time constant of the reward kernel was $\tau = 100$ms.

3.3.7.7 Details to computer simulation 5: Training a readout neuron with reward-modulated STDP to recognize isolated spoken digits

Spike representations of speech utterances. The speech utterances were preprocessed by the cochlea model described in Lyon (1982), which captures the filtering properties of the cochlea and hair cells in the human inner ear. The resulting analog signals were encoded by spikes with the BSA spike encoding algorithm described in Schrauwen and Campenhout (2003). We used the same preprocessing to generate the spikes as in Verstraeten et al. (2005). The spike representations had a duration of about 400 ms and 20 input channels. The input channels were connected topographically to the cortical microcircuit model. The neurons in the circuit were split into 20 disjunct subsets of 27 neurons, and each input channel was connected to the 27 neurons in its corresponding subsets. The readout neuron was trained with 20 different spike inputs to the circuit, where 10 of them resulted from utterances of digit "one", and the other 10 resulted from utterances of digit "two" by the same speaker.

Training procedure. We performed 2000 training trials, where for each trial a spike representation of a randomly chosen utterance out of 10 utterances for one digit was injected into the circuit. The digit changed from trial to trial. Whenever the readout neuron spiked during the presentation of an utterance of digit "two", a positive pulse was generated in the reward signal, and accordingly, for utterances of digit "one", a negative pulse in the reward was generated. We used the reward signal from equation (3.16). The amplitudes of the positive and negative pulses were $\alpha_P = -\alpha_N = 0.883$. The time constant of the reward kernel $\varepsilon_r(r)$ was $\tau = 100$ms. The pulses in the reward were delayed $d_r = 300$ ms from the spikes that caused them.

Cortical microcircuit details. The cortical microcircuit model consisted of 540 neurons with twenty percent of the neurons randomly chosen to be inhibitory, and the others excitatory. The recurrent connections in the circuit were created randomly with a connection probability of 0.1. Long-term plasticity was not modeled in the circuit synapses.

The synapses for the connections from the input neurons to the circuit neurons were static, current based with axon conduction delay of 1ms, and exponentially decaying PSR with time constant $\tau_e = 3$ ms and amplitude $w_{input} = 0.715$ nA.

3.4 Discussion

We have presented in this work analytical tools which make it possible to predict under which conditions reward-modulated STDP will achieve a given learning goal in a network of neurons. These conditions specify relationships between parameters and auxiliary functions (learning curves for STDP, eligibility traces, reward signals etc.) that are involved in the specification of the reward-modulated STDP learning rule. Although our analytical results are based on some simplifying assumptions, we have shown that they predict quite well the outcomes of computer simulations of quite complex models for cortical networks of neurons.

We have applied this learning theory for reward-modulated STDP to a number of biologically relevant learning tasks. We have shown that the biofeedback result of Fetz and Baker Fetz and Baker (1973) can in principle be explained on the basis of reward-modulated STDP. The underlying credit assignment problem was extremely difficult, since the monkey brain had no direct information about the identity of the neuron whose firing rate was relevant for receiving rewards. This credit assignment problem is even more difficult from the perspective of a single synapse, and hence for the application of a local synaptic plasticity rule such as reward-modulated STDP. However our theoretical analysis (see equation (3.10), (3.11)) has shown that the longterm evolution of synaptic weights depended only on the correlation of pairs of pre- and postsynaptic spikes with the reward signal. Therefore the firing rate of the rewarded neuron increased (for a computer simulation of a recurrent network consisting of 4000 conductance based LIF neurons with realistic background noise typical for in-vivo conditions, and 228954 synapses that exhibited data-based short term synaptic plasticity) within a few minutes of simulated biological time, like in the experimental data of Fetz and Baker (1973), whereas the firing rates of the other neurons remained invariant (see Fig. 3.3B). We were also able to model differential reinforcement of two neurons in this way (Fig. 3.4). These computer simulations demonstrated a remarkable stability of the network dynamics (see Fig. 3.3A, 3.4A, 3.5) in spite of the fact that all excitatory synapses were continuously subjected to reward-modulated STDP. In particular, the circuit remained in the asynchronous irregular firing regime, that resembles spontaneous firing activity in the cortex Brunel (2000). Other STDP-rules (without reward modulation) that maintain this firing regime have previously been exhibited in Morrison et al. (2007).

Whereas this learning task focused on firing rates, we have also shown (see Fig. 3.7) that neurons can learn via reward-modulated STDP to respond to inputs with particular spike trains, i.e., particular temporal output patterns. It has been pointed out in Farries and Fairhall (2007) that this is a particularly difficult learning task for reward-modulated STDP, and it was shown there that it can be accomplished with a modified STDP rule and more complex reward prediction signals without delays. We have complemented the results of Farries and Fairhall (2007) by deriving specific conditions (equation (3.13)-(3.15)) under which this learning task can be solved by the standard version of reward-modulated STDP. Extensive computer simulations have shown that these analytically derived conditions for a simpler neuron model predict also for a LIF neuron with conductance based synapses whether it is able to solve this learning task. Fig. 3.8 shows that this learning theory for reward-modulated STDP is also able to predict quite well *how fast* a neuron can learn to produce a desired temporal output pattern. An interesting aspect of Farries and Fairhall (2007) is that there also the utility of third signals that provide information about changes in the expectation of reward was explored. We have considered in this work only learning scenarios where reward prediction is not possible. A logical next step will be to extend our learning theory for reward-modulated STDP to scenarios from classical reinforcement learning theory that include reward prediction.

We have also addressed the question to what extent neurons can learn via reward-modulated STDP to respond with different firing rates to different spatio-temporal presynaptic firing patterns. It had already been shown in Izhikevich (2007) that this learning rule enables neurons to classify spatial firing patterns. We have complemented this work by deriving an analytic expression for the expected weight change in this learning scenario (see equation (3.17)), which clarifies to what extent a neuron can learn by reward-modulated STDP to distinguish differences in the temporal structure of presynaptic firing patterns. This theoretical analysis showed that in the extreme case, where all incoming information is encoded in the relative timing of presynaptic spikes, reward-modulated STDP is not able to produce a higher average membrane potential for selected presynaptic firing patterns, even if that would be rewarded. But it is able to increase the variance of the membrane potential, and thereby also the number of spikes of any neuron model that has (unlike the simple linear Poisson neuron) a firing threshold. The simulation results in Fig. 3.9 confirm that in this way a LIF neuron can learn with the standard version of reward-modulated STDP to discriminate even purely temporal presynaptic firing patterns, by producing more spikes in response to one of these patterns.

A surprising feature is, that although the neuron was rewarded here only for responding with a higher firing rate to one presynaptic firing pattern P, it automatically started to respond to this pattern P with a specific temporal spike pattern, that advanced in time during training (see Fig. 3.9A).

Finally, we have shown that a spiking neuron can be trained by reward-modulated STDP to read out information from a simulated cortical microcircuit (see Fig. 3.10). This is insofar of interest, as previous work Maass et al. (2002b); Häusler and Maass (2007); Maass et al. (2007) had shown that models of generic cortical microcircuits have inherent capabilities to serve as preprocessors for such readout neurons, by combining in diverse linear and nonlinear ways information that was contained in different time segments of spike inputs to the circuit ("liquid computing model"). The classification of spoken words (that were first transformed into spike trains) had been introduced as a common benchmark task for the evaluation of different approaches towards computing with spiking neurons Hopfield and Brody (2001); Maass et al. (2002b, 2004); Destexhe and Marder (2004); Verstraeten et al. (2005). But so far all approaches that were based on learning (rather than on clever constructions) had to rely on supervised training of a simple linear readout. This gave rise to the question whether also biologically more realistic models for readout neurons can be trained through a biologically more plausible learning scenario to classify spoken words. The results of Fig. 3.10 may be interpreted as a tentative positive answer to this question. We have demonstrated that LIF neurons with conductance based synapses (that are subject to biologically realistic short term plasticity) can learn without a supervisor through reward-modulated STDP to classify spoken digits. In contrast to the result of Fig. 3.9, the output code that emerged here was a rate code. This can be explained through the significant in-class variance of circuit responses to different utterances of the same word (see Fig. 3.10C, D). Although the LIF neuron learnt here without a supervisor to respond with different firing rates to utterances of different words by the same speaker (whereas the rate output was very similar for both words at the beginning of learning, see Fig. 3.10E), the classification capability of these neurons has not yet reached the level of linear readouts that are trained by a supervisor (for example, speaker independent word classification could not yet be achieved in this way). Further work is needed to test whether the classification capability of LIF readout neurons can be improved through additional preprocessing in the cortical microcircuit model, through a suitable variation of the reward-modulated STDP rule, or through a different learning scenario (mimicking for example preceding developmental learning that also modifies the presynaptic circuit).

The new learning theory for reward-modulated STDP will also be useful for biological experiments that aim at the clarification of details of the biological implementation of synaptic plasticity in different parts of the brain, since it allows to make predictions which types and time courses of signals would be optimal for a particular range of learning tasks. For each of the previously discussed learning tasks, the theoretical analysis provided conditions on the structure of the reward signal $d(t)$ which guaranteed successful learning. For example, in the biofeedback learning scenario (Fig. 3.3), every action potential of the reinforced neuron led – after some delay – to a change of the reward signal $d(t)$. The shape of this change was defined by the reward kernel $\varepsilon(r)$. Our analysis revealed that this reward kernel can be chosen rather arbitrarily as long as the integral over the kernel is zero, and the integral over the product of the kernel and the eligibility function is positive. For another learning scenario, where the goal was that the output spike train S_j^{post} of some neuron j approximates the spike timings of some target spike train S^* (Fig. 3.7), the reward signal has to depend on both, S_j^{post} and S^*. The dependence of the reward signal on these spike timings was defined by a reward kernel $\kappa(r)$. Our analysis showed that the reward kernel has to be chosen for this task so that the synapses receive positive rewards if the postsynaptic neuron fires close to the time of a spike in the target spike train S^* or somewhat later, and negative rewards when an output spike occurs in the order of ten milliseconds too early. In the pattern discrimination task of Fig. 3.9 each postsynaptic action potential was followed – after some delay – by a change of the reward signal which depended on the pattern presented. Our theoretical analysis predicted that this learning task can be solved if the integrals A_i^P and A_i^N defined by equation (3.18) are such that $A_i^P > 0$ and $A_i^N \approx -A_i^P$. Again, this constraints are fulfilled for a large class of reward kernels, and a natural choice is to use a non-negative reward kernel ε_r. There are currently no data available on the shape of reward kernels in biological neural systems. The previous sketched theoretical analysis makes specific prediction for the shape of reward kernels (depending on the type of learning task in which a biological neural system is involved) which can potentially be tested through biological experiments.

An interesting general aspect of the learning theory that we have presented in this work is that it requires substantial trial-to-trial variability in the neural circuit, which is often viewed as "noise" of imperfect biological implementations of theoretically ideal circuits of neurons. This learning theory for reward-modulated STDP suggests that the main functional role of noise is to maintain a suitable level of spontaneous firing (since if a neuron does not fire, it cannot find out whether this will be rewarded), which should vary from trial to trial in order to explore which firing patterns are rewarded.[5] On the other hand if a neuron fires primarily on the basis of a noise current that is directly injected into that neuron, and not on the basis of presynaptic activity, then STDP does not have the required effect on the synaptic connections to this neuron (see Fig. S6). This perspective opens the door for subsequent studies that compare for concrete biological learning tasks the theoretically derived optimal amount and distribution of trial-to-trial variability with corresponding experimental data.

[5]It had been shown in Maass et al. (2002b); Häusler and Maass (2007); Maass et al. (2007) that such highly variable circuit activity is compatible with a stable performance of linear readouts.

3.4.1 Related Work

The theoretical analysis of this model is directly applicable to the learning rule considered in Izhikevich (2007). There, the network behavior of reward-modulated STDP was also studied some situations different from the ones in this work. The computer simulations of Izhikevich (2007) operate apparently in a different dynamic regime, where LTD dominates LTP in the STDP-rule, and most weights (except those that are actively increased through reward-modulated STDP) have values close to 0 (see Fig. 1b and d in Izhikevich (2007), and compare with Fig. 3.5 in this chapter). This setup is likely to require for successful learning a larger dominance of pre-before-post over post-before-pre pairs than the one shown in Fig. 3.3E. Furthermore, whereas a very low spontaneous firing rate of 1 Hz was required in Izhikevich (2007), computer simulation 1 shows that reinforcement learning is also feasible at spontaneous firing rates which correspond to those reported in Fetz and Baker (1973) (the preceding theoretical analysis had already suggested that the success of the model does not depend on particularly low firing rates). The articles Baras and Meir (2007) and Florian (2007) investigate variations of reward-modulated STDP rules that do not employ learning curves for STDP that are based on experimental data, but modified curves that arise in the context of a very interesting top-down theoretical approach (distributed reinforcement learning Baxter and Bartlett (1999)). The authors of Pfister et al. (2006) arrive at similar learning rules in a supervised scenario which can be reinterpreted in the context of reinforcement learning. We expect that a similar theory as we have presented in this work for the more commonly discussed version of STDP can also be applied to their modified STDP rules, thereby making it possible to predict under which conditions their learning rules will succeed. Another reward based learning rule for spiking neurons was recently presented in Fiete and Seung (2006). This rule exploits correlations of a reward signal with noisy perturbations of the neuronal membrane conductance in order to optimize some objective function. One crucial assumption of this approach is that the synaptic plasticity mechanism "knows" which contributions to the membrane potential arise from synaptic inputs, and which contributions are due to internal noise. Such explicit knowledge of the noise signal is not needed in the reward-modulated STDP rule of Izhikevich (2007), which we have considered in this work. The price one has to pay for this potential gain in biological realism is a reduced generality of the learning capabilities. While the learning rule in Fiete and Seung (2006) approximates gradient ascent on the objective function, this cannot be stated for reward-modulated STDP at present. Timing-based pattern discrimination with a spiking neuron, as discussed in the section "Pattern discrimination with reward-modulated STDP" of this work, was recently tackled in Gütig and Sompolinsky (2006). The authors proposed the tempotron learning rule, which increases the peak membrane voltage for one class of input patterns (if no spike occurred in response to the input pattern) while decreasing the peak membrane voltage for another class of input patterns (if a spike occurred in response to the pattern). The main difference between this learning rule and reward-modulated STDP is that the tempotron learning rule is sensitive to the peak membrane voltage, whereas reward-modulated STDP is sensitive to local fluctuations of the membrane voltage. Since the time of the maximal membrane voltage has to be determined for each pattern by the synaptic plasticity mechanism, the basic tempotron rule is perhaps not biologically realistic. Therefore, an approximate and potentially biologically more realistic learning rule was proposed in Gütig and Sompolinsky (2006), where plasticity following error trials is induced at synapse i only if the voltage within the postsynaptic integration time after their activation exceeds a plasticity threshold κ. One potential problem of this rule is the plasticity threshold κ, since a good choice of this parameter strongly depends on the mean membrane voltage after input spikes. This problem is circumvented by reward-modulated STDP, which considers instead the local change in the membrane voltage. Further work is needed to compare the advantages and disadvantages of these different approaches.

3.4.2 Conclusion

Reward-modulated STDP is a very promising candidate for a synaptic plasticity rule that is able to orchestrate local synaptic modifications in such a way that particular functional properties of larger networks of neurons can be achieved and maintained (we refer to Izhikevich (2007) and Farries and Fairhall (2007) for discussion of potential biological implementations of this plasticity rule). We have provided in this work analytical tools which make it possible to evaluate this rule and variations of this rule not just through computer simulations, but through theoretical analysis. In particular we have shown that successful learning is only possible if certain relationships hold between the parameters that are involved. Some of these predicted relationships can be tested through biological experiments.

Provided that these relationships are satisfied, reward-modulated STDP turns out to be a powerful rule that can achieve self-organization of synaptic weights in large recurrent networks of neurons. In particular, it enables us to explain seemingly inexplicable experimental data on biofeedback in monkeys. In addition reward-modulated STDP enables neurons to distinguish complex firing patterns of presynaptic neurons, even for data-based standard forms of STDP, and without the need for a supervisor that tells the neuron when it should spike. Furthermore reward-modulated STDP requires substantial spontaneous activity and trial-to-trial variability in order to support successful learning, thereby providing a functional explanation for these ubiquitous features of cortical networks of neurons. In fact, not only spontaneous activity but also STDP itself may be seen in this context as a mechanism that supports the exploration of different firing chains within a recurrent network, until a solution is found that is rewarded because it supports a successful computational function of the network.

3.5 Acknowledgments

This chapter is based on the journal article *A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback* by Robert Legenstein[1] (RL), myself[1] (DP) and Wolfgang Maass (WM) (PLoS Computational Biology 4(10): e1000180, 2008). In this article RL contributed the theoretical analysis, RL, DP and WM conceived and designed the experiments and DP conducted the experiments and analysed the simulation results. RL, DP and WM wrote the manuscript.

Markus Diesmann, Eberhard Fetz, Razvan Florian, Yves Fregnac, Wulfram Gerstner, Nikos Logothetis, Abigail Morrison, Matthias Munk, Gordon Pipa and Dan Shulz contributed with helpful discussions during the preparation of the article. Malcolm Slaney provided a MATLAB implementation of the cochlea model from Lyon (1982). Benjamin Schrauwen, David Verstraeten, Michiel D'Haene and Stefan Klampfl provided additional code that used in the speech classification task (computer simulation 5).

[1]These authors contributed equally to the work in the paper.

Chapter 4

PCSIM: a Parallel Simulation Environment for Neural Circuits

Contents

The *P*arallel *C*ircuit *SIM*ulator (PCSIM) is a software package for simulation of neural circuits. It is primarily designed for distributed simulation of large scale networks of spiking point neurons. Although its computational core is written in C++, PCSIM's primary interface is implemented in the Python programming language, which is a powerful programming environment and allows the user to easily integrate the neural circuit simulator with data analysis and visualization tools to manage the full neural modeling life cycle. The main focus of this work is to describe PCSIM's full integration into Python and the benefits thereof. In particular we will investigate how the automatically generated bidirectional interface and PCSIM's object-oriented modular framework enable the user to adopt a hybrid modeling approach: using and extending PCSIM's functionality either employing pure Python or C++ and thus combining the advantages of both worlds. Furthermore, we describe several supplementary PCSIM packages written in pure Python and tailored towards setting up and analyzing neural simulations.

4.1 Introduction

Given the complex nonlinear nature of the dynamics of biological neural systems, many of their properties can be investigated only through computer simulations. The need of researchers to increase their productivity while implementing increasingly complex models without each time having to reinvent the wheel has become a driving force to develop simulators for neural systems that incorporate best known practices in simulation algorithms and technologies, and make it accessible to the user through a high-level user-friendly interface (Brette et al., 2007). It has also been brought to attention that it is of importance to use large neural networks with biologically realistic connectivity (on the order of 10^4 synapses per neuron) as simulation models of mammalian cortical networks (Morrison et al., 2005). Simulation of such large models can practically be done only by exploiting the computing power and the memory of multiple computers by means of a distributed simulation.

There are different neural simulation environments presently available and although many of them were initially envisioned for a specific purpose and domain of applicability, during continuing development their set of features expanded to improve generality and support construction of a wide range of different neural models; see (Brette et al., 2007) for a recent overview. The two most prominent tools are NEURON (Hines and Carnevale, 1997; Carnevale and Hines, 2006) and GENESIS (Bower and Beeman, 1998) which aim at simulation of detailed multi-compartmental neuron models and small networks of detailed neurons. Another class of quite general neural simulation environments which focus on the simulation of large-scale cortical network models and the improvement of their simulation efficiency through distributed computing include NEST (Gewaltig and Diesmann, 2007; Plesser et al., 2007), NCS (Brette et al., 2007) and SPLIT (Hammarlund and Ekeberg, 1998). There are also more dedicated neural simulation tools like iNVT (iLab Neuromorphic Vision Toolkit[1]) which is an example of a package specifically tailored for the domain of brain-inspired neuromorphic vision. All of the above simulation environments support parallel simulation of one model on multiple processing nodes by using commodity clusters and many of them can also be run on super-computers. The simulation tool PCSIM described in this work is designed for simulating neural circuits with a support for distributed simulation of large scale neural networks. Its development started as an effort to redesign the previous CSIM simulator[2] (Natschläger et al., 2003) and augment its capabilities, with the major extension being the implementation of a distributed simulation engine in C++ and a new convenient programming interface. The aim was to provide a general extensible framework for simulation of hybrid neural models that include both spiking and analog neural network components together with other abstract processing elements while making the setup and control of parallel simulations as convenient as possible for the user. Hence, given its current set of features, the PCSIM simulator is closest to the second group (NEST, NCS, SPLIT) of neural simulation environments mentioned above.

Performing a neural network simulation usually requires combined usage of several additional software tools together with the simulator, for stimulus preparation, analysis of output data and visualization. Being able to steer all the necessary tools from one programming environment reduces the complexity of setting up simulation

[1] http://ilab.usc.edu/toolkit/home.shtml
[2] http://www.lsm.tugraz.at/csim

experiments since all development can be done in a single programming language and the burden of developing utilities for conversion of data formats between heterogeneous tools is avoided. Given its object-oriented capabilities and its strong support for integration with other programming languages, the Python programming language is a very promising candidate for providing such a unifying software environment for simultaneous use of various scientific software libraries. As Python is becoming increasingly popular in the scientific community as an interpreting language of choice for scientific applications, the developers of many neural simulator tools decided to provide a Python interface for their simulator in addition to its legacy interface in a custom scripting language. Moreover, a simulation tool called Brian which uses Python as an implementation language was recently developed to bring to the user the full flexibility of an interpreting language in specifying and manipulating neural models (Goodman and Brette, 2008).

In spite of the evident practical advantages in using Python as the single programming language for all tasks during a neural modeling life cycle, there is the apparent discrepancy between the need for computational performance of the simulation and construction of the model on one hand, and rapid development of the model on the other. Using C++ can solve the performance issue, but will decrease the productivity of the modeler and requires higher level of programming skills and experience. In contrast Python is easy to learn, flexible to use and significantly increases the productivity of the modeler, however it lags far behind C++ in performance.[3] Hence, instead of adopting a single language, an alternative is to enable an easy mix and match of both languages during the development of a model, i.e. to introduce a *hybrid modeling approach* (Abrahams and Grosse-Kunstleve, 2003).

In this chapter we will describe how the modular object-oriented framework of PCSIM in combination with an automated interface generation supports such a hybrid modeling approach.

In particular, we briefly review PCSIM's main features (Sec. 4.2) before we describe the automated process to generate the Python interface (Sec. 4.3). In Sec. 4.4 we detail PCSIM's network construction application programming interface (API), which is a central part of PCSIM's object-oriented modular framework. In Sec. 4.5 we demonstrate another advantage of the hybrid modeling approach: we show how PCSIM's concept of a general network element can be used as an interface to another simulation tool. While these examples concentrate on the Python aspect of the hybrid modeling, we show in Sec. 4.6 how the user can easily extend PCSIM's functionality using C++. Additional PCSIM packages implemented in Python are reviewed in section 4.7. In Sec. 4.8 we discuss and summarize the presented concepts and approaches.

[3]The simulation tool Brian mentioned above, heavily uses the numerical Python package numpy (Oliphant, 2007) written in C to achieve reasonable performance.

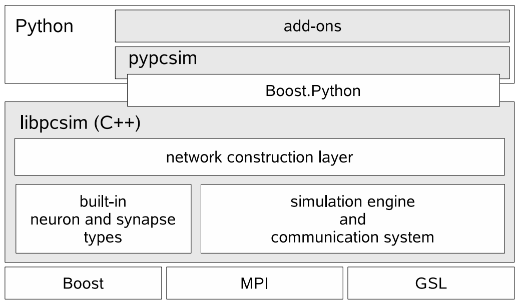
Python
add-ons
pypcsim
Boost.Python
libpcsim (C++)
network construction layer
built-in neuron and synapse types
simulation engine and communication system
Boost
MPI
GSL

Figure 4.1: Architecture overview of PCSIM

4.2 Overview

4.2.1 Architecture

The high-level architecture of PCSIM is depicted in Fig. 4.1. The PCSIM library written in C++ (libpcsim) constitutes the core of the simulator. The API of the PCSIM library is exposed to the Python programming language by means of the Python extension module pypcsim (see Sec. 4.3 for details). The library is made up of three main components: the simulation engine with its communication system, a pool of built-in network elements (i.e. neuron and synapse types) and the network construction layer. Before presenting the network construction layer in detail in Sec. 4.4 we will briefly describe in the next paragraphs the main features of the underlying simulation engine and its communication system.

The simulation engine integrates all the network elements (typically neurons and synapses) and advances the simulation to the next time step, and uses its communication system to handle the routing and delivery of *discrete and analog messages* (i.e. spikes and e.g. firing rates or membrane voltages) between the connected network elements. PCSIM's simulation engine is capable of running distributed simulations where the individual network elements are located at different computing nodes. Setting up a distributed simulation is handled easily from a users point of view: there are no (or very little) code changes necessary when switching from a non-distributed to a distributed simulation. The distributed simulation mode is intended for employing a cluster of machines for simulation of one large network where each machine integrates the equations of a subset of neurons and synapses in the network. A distributed PCSIM simulation runs as an MPI[4] based application composed of multiple MPI processes located on different machines[5]. The implementation of the spike routing, transfer and delivery algorithm between the nodes in a distributed simulation is based on the ideas presented in (Morrison et al., 2005). In addition PCSIM offers the possibility to run a simulation as a multi-threaded application, both in a non-distributed and a distributed setup. The multi-threaded mode is in-

[4] http://www-unix.mcs.anl.gov/mpi/

[5] To be precise, we use the C++ bindings offered by the MPICH2 library, where currently none of the advanced features of the MPI-2 standard are used.

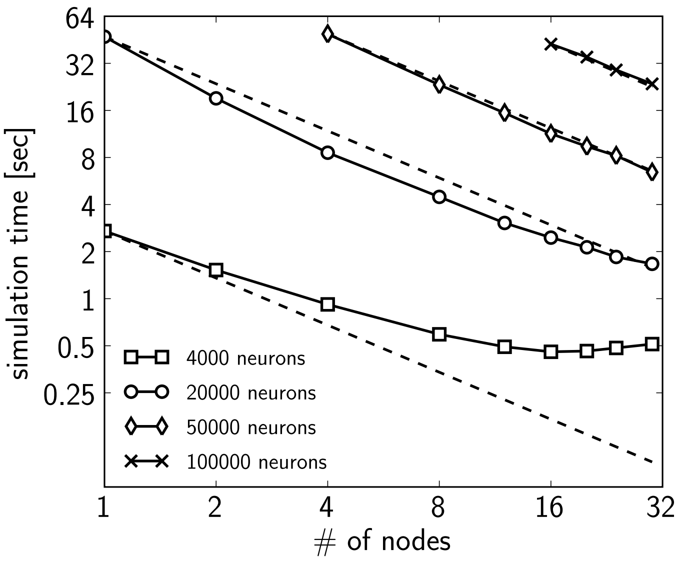

Figure 4.2: Simulation times of the CUBA network distributed over different number of processing nodes, compared to the expected simulation time (dashed line) (see text for details). Four different sizes of networks were simulated: 4000 neurons with on average 1.6×10^6 synapses (diamonds), 20000 neurons with on average 40×10^6 synapses (stars), 50000 neurons with on average 250×10^6 synapse (circles) and 100000 neurons with on average 1×10^9 synapses (squares). The plotted simulation times are averages over 12 simulation runs. The variation of simulation time between different simulation runs was small, therefore we did not show it.

tended for performing simulations on one multi-processor machine when one wants to split the computational workload among multiple threads in one process, each running on a different processor. However, we should note that the multi-threaded simulation engine is still undergoing optimization, as we are working on improvement of the scaling of the multi-threaded simulation to match the scaling achieved with an equivalent distributed simulation.

4.2.2 Scalability and Domain of Applicability

One of the goals of the development of PCSIM was enabling simulations of large neural networks on standard computer clusters through distributed computing. By utilizing the parallel capabilities of PCSIM the simulation time for a model can be reduced by using more processors (on multiple machines) as computing resources.

As a test of the scalability, we performed multiple simulations with the PCSIM implementation of the CUBA model described in (Brette et al., 2007), with different number of leaky integrate-and-fire neurons (4000, 20000, 50000 and 100000) and distributed over a different number of processors (each processor on a different machine). We changed the resting potential in the neuron equations from $-49mV$ to $-60\,mV$ such that the network does not show any spontaneous activity. In order to elicit a spiking activity in the network, an input neuron population of 1000 neurons was connected randomly to it with probability 0.1, i.e. each neuron in the network receives inputs from on average 100 input neurons. The input neurons fired homogeneous Poisson spike trains at a rate of 5 Hz. The simulation was performed for 1 sec biological time with a time step of 0.1 ms. We have set the connection probability within the network to 0.1, in order to reach realistic number of 10000 synapses per neuron for the network size of 100000 neurons. The transmission delay of spikes was set to 1 ms. We scaled the weights of the network so that the mean firing rate of the neurons was between 2.4 and 2.7 Hz for all network sizes (more precisely 2.68 Hz, 2.55 Hz, 2.52 Hz and 2.45 Hz for the network with 4000 , 20000, 50000 and 10000 neurons, respectively).

The used machines had Intel® Xeon^{TM}64 bit CPUs with 2.66 GHz and 4 MB level-2 processor cache, and 8 GB of RAM. They were connected in a 1 Gbit/s Ethernet LAN.

If we assume ideal linear speed-up, then the expected simulation time of a model on N machines given the actual simulation time on K machines is equal to the simulation time on K machines times K divided by N. In the evaluation of the scaling, for the estimation of the expected simulation time (see Fig. 4.2) we used the measured simulation time of the model on the minimum number of machines used for that particular network size. Namely, we used the actual simulation time on $K = 1$ machine for the network sizes of 4000 and 20000 neurons, and the simulation time on $K = 4$ and $K = 16$ machines for the network sizes of 50000 and 100000 neurons respectively.

Fig. 4.2 shows that in the case of 4000 neurons the computational load on each node is quite low, hence the cost of the spike message passing dominates the simulation time which results in sub-linear scaling. For the networks with 20000 and 50000 neurons the actual simulation time is shorter than the expected simulation time indicating a supra-linear speed-up for up to 24 nodes. For more than 24 nodes the actual simulation time approaches the expected simulation time. The reason for the supra-linear speed-up is more efficient usage of the processor cache when the network is distributed over larger number of nodes (Morrison et al. (2005)). For the network with 100000 neurons the speed-up is not distinguishable from the expected linear speed-up (taking $K = 16$ nodes as the base measurement).

The combination of features that PCSIM supports makes it suitable for various types of neural models. Its domain of applicability can be considered across two complementary aspects: the size of networks that can be simulated, and the variety of different models that can be constructed and simulated, determined by the available neuron and synapse models, plasticity mechanisms, construction algorithms and similar. Concerning the size of models, because of its distributed capabilities PCSIM is mainly targeted towards large neural systems with realistic cortical connectivity composed of 10^5 neurons and above. As the results from the scalability test show, a spiking network with 10^5 neurons and 10^4 synapses per neuron can be simulated in a reasonable time on a commodity cluster with about 20 machines, and the speed-up is linear when more machines are employed for the simulation.

Regarding the support for construction of various different models in PCSIM, the generality of the communication system and the extensibility with custom network elements enables simulation of hybrid models (spiking and analog networks) incorporating different levels of abstraction. By utilizing the construction framework also structured models with diversity of neuron and synapse types and varying parameter values can be defined and simulated, and the built-in support for synaptic plasticity further expands the domain of usability towards models that investigate synaptic plasticity mechanisms.

4.3 Python interface generation

In order to enable a hybrid modeling approach we wanted to use a Python interface generation tool that was capable of wrapping PCSIM's object-oriented and modular API such that the Python API will be as close as possible to the C++ API. Our choice for this purpose was the Boost.Python[6] library (Abrahams and Grosse-Kunstleve, 2003). The strength of Boost.Python is that by using advanced C++ compile-time introspection and template meta-programming techniques it provides comprehensive mappings between C++ and Python constructs and idioms. There is support, amongst others, for exception handling, iterators, operator overloading, standard template library (STL) containers and Python collections, smart pointers and virtual functions that can be overridden in Python. The later feature makes the interface bidirectional, meaning that in addition to the possibility of calling C++ code from Python, user extension classes implemented in Python can be called from within the C++ framework. This is an enabler for the targeted hybrid modeling approach; we will see examples for this later on in this chapter.

However, using Boost.Python without any additional tools does not lead to a solution where the interface can be generated in an automatic fashion since for each new class added to the library's API one would have to write a substantial piece of Boost.Python code. As automatic Python wrapping of the C++ interface is one of the main prerequisites for leveraging a hybrid modeling approach, a solution is needed to automatically synchronize the Python and C++ API of a library like libpcsim. Fortunately, there exists the Py++ package[7] which was developed to alleviate the repetitive process of writing and maintaining Boost.Python code. Py++ by itself is an object-oriented framework for creating custom Boost.Python code generators for an application library written in C++. It builds on GCC-XML[8], a C++ parser based on the GCC compiler that outputs an XML representation of the C++ code. Py++ uses this structured information together with some user input, in form of a Python program, and produces the necessary Boost.Python code, constituting the Python interface for a specified set of C++ classes and functions (see Fig. 4.3).

[6]`http://www.boost.org/doc/libs/release/libs/python/doc/`
[7]`http://www.language-binding.net/`
[8]`http://www.gccxml.org`

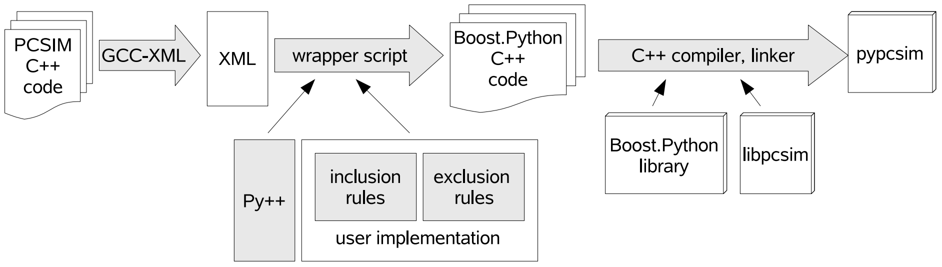

Figure 4.3: The processing steps in the generation of the Python interface for PCSIM.

Finally the Boost.Python C++ code is compiled and linked together with the C++ library under consideration (libpcsim in our case) to produce the Python extension module containing the Python API of the library (pypcsim in our case). Thus, the work of the developer (and the user as we will see later on) reduces to a definition of high-level rules to select which classes and methods should be exposed.

For the generation of the PCSIM Python interface pypcsim, we split the rules Py++ needs into two subsets, inclusion and exclusion rules (see Fig. 4.3). The inclusion rules contain the rules that mark a selected set of classes to be exposed to Python. The exclusion rules contain the post-processing, where some of the methods of the classes that were included in the inclusion rules are marked to be excluded, and call policies are defined for the included methods that require them[9]. Py++ allows to specify the rules in a high-level, generic fashion, making them robust to changes in the interface of the PCSIM C++ library. Hence, in most cases changes in the PCSIM API did not require changes in the Python program that generates the wrapper code, which simplified its maintenance. An example of such a high-level rule would be "In all classes that are derived from class A, do not expose the method that returns a pointer of type B". Such a general rule will then be still valid if for example we introduce more classes derived from A, or add additional functions that return a pointer of type B in some of the classes.

To summarize, the Python integration of PCSIM using Boost.Python together with the Py++ code generator allowed us to come up with a solution to automatically expose PCSIM's object-oriented and modular API bidirectionally in Python. In the following sections we will show how such an bidirectional integration of PCSIM into Python can practically be used and which possibilities and advantages arise.

4.4 Network construction

A large portion of the Python PCSIM interface is devoted to the construction of neural circuits. At the lowest level PCSIM provides methods to create individual

[9]Call policies define the change of ownership of objects that cross the boundaries of the C++ library, i.e. the object passed from Python to the C++ library and from the C++ library to Python.

network elements (i.e. neurons and synapses) and to connect them together.

On top of these primitives a powerful and extensible framework for circuit construction based on probabilistic rules is built. The source of inspiration for the interface of the framework was the Circuit Tool in the CSIM simulator[10] and PyNN, an API for simulator-independent procedural definition of spiking neural networks (Davison et al., 2008). We will use a concrete example[11], described in more depth in the next subsection, to present the network construction framework and its typical use cases where emphasis is put on those features that were enabled by the bidirectional Python interface generated by the approach described in Sec.4.3.

4.4.1 The example model

We selected the model to be simple enough for didactic reasons, but complete enough with all the elements necessary to explain the main novel concepts of the interface and its Python extensibility features. The connectivity patterns are based on experimental data that we use in our current research work. The model consists of a spatial population of neurons located on a 3D grid with integer coordinates within a volume of $20 \times 20 \times 6$. 80% of the neurons in the model are excitatory, and the rest are inhibitory. The excitatory neurons are modeled as regular spiking and the inhibitory neurons as fast spiking Izhikevich neurons (Izhikevich, 2004). The connections between excitatory neurons in the network are created according to the trivariate probabilistic model defined in (Buzas et al., 2006). This connectivity model describes the distribution of the excitatory patchy long-range lateral connections found in the superficial layers of the primary visual cortex in cats that depends on the lateral distance of the cells and their orientation preference. Orientation preference is the affinity of V1 cells to fire more when a bar with a specific orientation angle is present in their receptive fields. The connectivity rule is defined by the following equations that express the connectivity probability between two excitatory cells.

$$P(\mathbf{l_i}, \mathbf{l_j}, \phi_i, \phi_j) = C\, G(\mathbf{l_i}, \mathbf{l_j})\, V(\phi_i, \phi_j) \tag{4.1}$$

$$G(\mathbf{l_i}, \mathbf{l_j}) = e^{-\frac{|\mathbf{l_i} - \mathbf{l_j}|^2}{2\sigma^2}} \tag{4.2}$$

$$V(\phi_i, \phi_j) = e^{\kappa \cos 2(\phi_i - \phi_j)} \tag{4.3}$$

$\mathbf{l_i} = (x_i, y_i)$ and $\mathbf{l_j} = (x_j, y_j)$ are the 2D locations and ϕ_i and ϕ_j are the orientation preferences of the pre- and postsynaptic neurons i and j. The function G introduces the dependence of the connectivity probability on the lateral distance between the neurons, and V models the dependency on the differences in the orientation preferences of the neurons. C, κ and σ are scaling coefficients. The values for the preferred orientation angles of the neurons in the example are generated by evolving a self-organizing map (SOM) (Obermayer and Blasdel, 1993). Additionally

[10]http://www.lsm.tugraz.at/circuits

[11]The full source code of this example is available in the supplementary material.

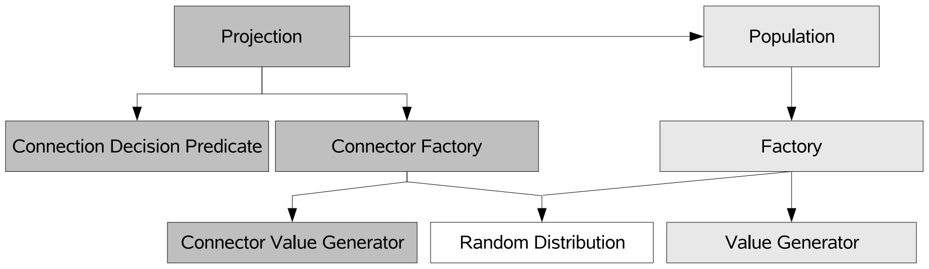

Figure 4.4: A diagram of the most important concepts within the network construction interface. The arrows indicate a "uses" relationship between the concepts.

the conduction delay of a connection between excitatory neurons is probabilistically dependent on the distance between the 3D locations of its pre- and postsynaptic neurons.

$$D(\mathbf{l_i}, \mathbf{l_j}) = D_0 \frac{|\mathbf{l_i} - \mathbf{l_j}|}{N(\mu, \sigma, b_l, b_u)} \tag{4.4}$$

Here $N(\mu, \sigma, b_l, b_u)$ is a bounded normal distribution representing the transmission velocity of the axon. A random value from $N(\mu, \sigma, b_l, b_u)$ is sampled as follows: first a random number from a normal distribution with mean μ and standard deviation σ is drawn and if that value is not within the range $[b_l, b_u]$, then it is drawn from an uniform distribution with that range. D_0 represents a proper scaling factor in the formula.

4.4.2 The framework: object-oriented, modular and extensible

Fig. 4.4 shows the basic concepts of PCSIM's construction framework together with their interactions during the construction process. This framework allows model specification in terms of *populations* of neurons connected by probabilistically defined connectivity patterns called *projections*.

A population of network elements utilizes several object *factories* to generate the network elements. A factory encapsulates the logic for the neuron and synapse generation decoupled from the other parts of the construction process. Every time a new neuron is to be created in a population the factory is used to generate the neuron object. The object factories can use either *random distribution* objects or *value generators* to generate values for the *parameters* and *attributes* of the network element instances. When we talk about a parameter we mean a parameter of the differential equations used to model a neuron or synapse. In contrast an attribute describes any other (more abstract) property of a network element. In our example the orientation preference ϕ will be such an attribute of an excitatory neuron.

A projection manages connections between two populations. During the construction phase of a projection a *connection decision predicate* is used to determine whether a connection should be created for a pair of neurons. A *connector factory* is then used to create instances of the connector elements like synapses (this is analogous to the object factory for populations). The connector factory also uses *random distributions* or *connector value generators* for the parameter values of the connector elements. In order to implement a specific construction algorithm, the user typically just needs to implement custom *value generator* and *connection decision predicate* classes, as we will demonstrate in the following subsections.

4.4.3 Factories: creating network elements from models

We will start constructing the network model by defining the classes (or families) of neurons models: inhibitory and excitatory neurons. This is accomplished by defining an element factory for each family. As explained in Sec. 4.4.1 the excitatory neurons have an orientation preference ϕ which depends on the location of the neuron in the population. For this reason we will associate the attribute `phi` with each excitatory neuron:

```
exc_factory = Factory( model   = IzhiNeuron( type = "RS" ),
                       Vinit   = UniformDistribution( -50e-3, -60e-3 ),
                       attribs = dict( phi = OrientationPreferValGen() )
```

The statement above creates a factory for the excitatory family of neurons based on a regular spiking (RS) Izhikevich neuron model (Izhikevich, 2004) where `IzhiNeuron` is a built-in network element class. The keyword argument `Vinit = UniformDistribution(...)` associates a uniform random number generator with the initial membrane voltage `Vinit`. This has the effect that whenever the factory is used to generate an actual instance of an excitatory neuron, the parameter `Vinit` will be randomly chosen from the interval $[-50, -60]$ mV. Finally the keyword argument `attribs = dict( phi = ... )` has two effects: a) the attribute `phi` is attached to `exc_factory` and b) the custom *value generator* `OrientationPreferValGen` is used to generate a particular value for `phi` each time `exc_factory` is asked to generate an instance of an excitatory model neuron. The value of the `phi` attribute will be used afterwards for the creation of synaptic connections.

In the example we implement the custom value generator `OrientationPreferValGen` in pure Python. This is enabled by the particular feature of Boost.Python which allows C++ virtual functions to be overridden from within Python.

```
class OrientationPreferValGen(PyAttributePopObjectValueGenerator):

    def __init__(self):
        PyAttributePopObjectValueGenerator.__init__(self)
        self.map = som.OrientationMapSOM([20,20])

    def generate(self, rng):
        return self.map.pref( self.loc().x(), self.loc().y() )
```

Value generators (in this case to be derived from `PyAttributePopObjectValueGenerator`) have a simple interface composed of the constructor `__init__` and the method `generate` which have to be implemented by the user. In our particular example we create the orientation map, that maps 2D coordinates to an orientation preference angle in the constructor, and will use it in the method `generate`. The map is based on the SOM algorithm encapsulated in the Python class `OrientationMapSOM` (details not relevant here). The `generate` method is called to determine the value of the orientation angle attribute `phi` whenever a neuron instance from the factory has to be created. The value generator inherits several convenient methods from its base class that one can use for accessing properties of the neuron for which `generate` is called, like `self.loc` to get the 3D location of the neuron within a population (see next section). We then pass the x and y coordinates to the orientation map (method `pref`) in order to calculate the value of the orientation preference angle.

For the inhibitory neuron model we create a similar factory:

```
inh_factory = Factory( model   = IzhiNeuron( type = "FS" ),
                       Vinit   = UniformDistribution( -50e-3, -60e-3 ),
                       attribs = dict( ) )
```

The difference to the excitatory neuron model is that a fast spiking (FS) Izhikevich neuron model is used and the attribute dictionary `attribs = dict( )` is empty. This is because there is no orientation preference of the inhibitory cells in the considered model.

4.4.4 Neuron populations

A population in PCSIM represents an organized set of neurons that can be manipulated as one structural unit in the model. In the `AugmentedSpatialPopulation` that we will use in this example, the neurons have associated 3D coordinates, a family identifier, and an extensible set of custom attributes that the user can attach to each of the neurons. We already encountered this in the previous section. The family identifier allows the definition of multiple families/classes of neurons, i.e. subsets of neurons with similar properties, within a single population. Our population will have two families of neurons, the family of excitatory and the family of inhibitory neurons. For each of the two families of neurons we have specified in the previous section a factory that will be used to generate the neuron instances within the population.

```
pop = AugmentedSpatialPopulation( net, [ exc_factory(), inh_factory() ],
                                  RatioBasedFamilies( [ 4, 1 ] ),
                                  CuboidIntegerGrid3D( 20, 20, 6 ) )

exc_pop, inh_pop = pop.splitFamilies()
```

Note that the first argument (`net`) specifies the overall network to which this population of neurons will belong. The class `CuboidIntegerGrid3D`, which is a built-in specialization of the more general concept of an arbitrary set of points in 3D, defines the possible locations for the neurons (integer coordinates within a volume of $20 \times 20 \times 6$). The population is to be composed of two families of neurons (excitatory and inhibitory), created by the two given factories (`exc_factory` and `inh_factory`). To accomplish this we use a `RatioBasedFamilies` object which randomly chooses for each 3D location from which family of neurons the particular instance will be created. Specifying the ratio 4:1 for excitatory to inhibitory neurons yields the desired 80% excitatory neurons. The class `RatioBasedFamilies` is a built-in specialization of the general concept of a *spatial family identifier generator* which encapsulates the logic for deciding which factory to use depending on the 3D location.

For the purpose of more convenient setup of connections later on, the created population is split into two sub-populations, one for each family.

4.4.5 Projections: managing synaptic connections

The synaptic connections in the network construction interface are created by means of projections. A projection is a construct that represents a set of synaptic connections originating from one population of neurons and terminating at another population[12]. PCSIM has built-in construction algorithms for creating various types of connection projections, like constant probability random connectivity or random connectivity with probability dependent on the distance (or lateral distance) between the neurons.

However, to create a projection with a specific connectivity pattern, one usually defines a custom *connection decision predicate*. A decision predicate decides for an individual pair of neurons whether to form a connection based on the parameters and attributes of those neurons. In our example we implemented the connection decision predicate `OrientationSpecificConnPredicate` in pure Python, encapsulating the probabilistic rule for connection making from Equ. 4.1, which states that the connection probability depends on the distance between and the orientation preferences of the pre- and postsynaptic neurons.

```
class OrientationSpecificConnPredicate(
                          PyAugmentedConnectionDecisionPredicate):

  def __init__(self, C):
     PyAugmentedConnectionDecisionPredicate.__init__(self)
     self.orient_conn_prob = OrientationSpecConnProbability(C)
     self.unidist = UniformDistribution(0.0, 1.0)

  def decide(self, src, dst, rnd ):
     prob = self.orient_conn_prob(self.src_attr(src, 'phi'),
                                  self.dest_attr(dst, 'phi'),
                                  self.dist_2d( src, dst ) )
     return self.unidist(rnd) < prob
```

The `PyAugmentedConnectionDecisionPredicate` base class is used when one has to define a custom connection decision predicate that uses the neuron attributes

[12]The source and destination populations can be the same if the goal is to create recurrent connections in one population.

and connects neurons from populations of type `AugmentedSpatialPopulation`. To complete the implementation of the predicate, it is required to override the `decide` method and fill the constructor with the necessary initializations. The method `decide` is called within the connection construction process for each candidate pair of neurons that could be connected and is expected to output true (make a connection) or false (no connection). In our example, we create an instance (`orient_conn_prob`) of the `OrientationSpecConnProbability` class to calculate the probability according to the Equ. 4.1 (the full implementation of the class is available in the supplementary material). This instance is called in the `decide` method with the orientation preferences of the candidate source and destination neurons and their lateral distance as arguments. The orientation preferences are obtained via the `src_attr` and `dest_attr` methods (inherited from the base class), and the lateral distance via the `dist_2d` method. By comparing a uniformly distributed random number to the calculated probability a Bernoulli distribution with the desired probability for the outcome true is generated.

Before we can create the projection we have to define a connector factory (class `ConnFactory`) that will be used to generate the synapse objects within the projection.

```
ee_syn_factory =  ConnFactory(
                         model = StaticSpikingSynapse(W = 1e-4),
                         delay = DelayCond(v_mean = 2e2, v_SH = 0.2,
                                           v_min = 0.1e-3, v_max = 5e-3) )
```

The connector factory differs from the element factory objects used in conjunction with neuron populations, in that the parameters of the created objects (typically synapses) can depend on the attributes of the source and destination network elements they are connecting. In our example, the connector factory for the connections between excitatory neurons is based on a current-based synapse model with exponential decay post-synaptic response (class `StaticSpikingSynapse` in PCSIM). Additionally, the `DelayCond` value generator is associated to the delay parameter of the synapse, which produces distance dependent delay values according to Equ. 4.4. The `DelayCond` is a built-in value generator in PCSIM.

Now we can create the projection that will generate all recurrent connections between the excitatory neurons.

```
ee_proj = ConnectionsProjection( exc_pop, exc_pop, ee_syn_factory(),
                PredicateBasedConnections(
                           OrientationSpecificConnPredicate( 1.0 ) ) )
```

We specify in the constructor of the projection the connector factory for generation of the synapses and the `PredicateBasedConnections` class instance that iterates over all candidate pre- and postsynaptic neurons and delegates the decision whether to make a connection to the connection decision predicate `OrientationSpecificConnPredicate` given as an argument.

A connection decision predicate is typically used when in the probabilistic connectivity definition the probability that two neurons are connected depends on the attributes and parameters of the two neurons and is independent from the other created connections. In the general case, with such a connectivity, a separate decision whether to make a connection has to be made at each candidate neuron pair, yielding a complexity of the wiring algorithm that is quadratic with respect to the number of neurons. In a distributed scenario, a speed-up of the construction is possible by splitting the wiring workload among the multiple machines the model is simulated on. If the number of machines is increased with the number of neurons, keeping the number of neurons per node fixed, and if we assume that the number of input synapses per neuron does not increase, then the wiring time will scale linearly with the number of neurons.

For other connectivity schemes where further optimizations are possible, a faster wiring algorithm can be implemented directly in the class that iterates over the neuron pairs. For example, for the case of constant probability random connections, a special `RandomConnections` class that implements faster wiring can be used instead of `PredicateBasedConnections`. When using the `RandomConnections`, the wiring time is proportional to the number of created connections if the network is constructed on a single machine, and remains constant in the distributed case with the assumption that the number of machines is increased proportionally with the number of neurons.[13]

4.5 Custom network elements

The PCSIM communication system is general in a sense that it supports spiking and analog messages as communication between network elements. The network elements are not restricted to one type of message and can have multiple input and output ports, each of them capable of either receiving or sending spiking or analog messages (see Fig. 4.5A and Fig. 4.5B).

The generality of the framework allows the user to implement custom processing elements that map multiple inputs to multiple outputs and plug them in a network model inter-connected together with spiking or analog neural networks. Such custom network elements can either be implemented in C++ (see Sec. 4.6) or in pure Python. This feature of PCSIM has various potential uses. For example the user can implement new neuron types for a preliminary experiment in Python first, instead of directly implementing them in C++ (see Sec. 4.6). Another possible usage is to implement more abstract or complex elements like a whole population of spiking neurons in Python by using vectors from the numerical Python package `numpy` [14] (Oliphant, 2007) for step-by-step integration of the equations. This approach has been shown to have good performance, and is applicable for homogeneous neuron populations, where all neuron instances have the same neuron model (Brian

[13] It is out of scope of this work to detail the algorithms behind the efficient implementation of the network construction framework in the distributed simulation scenario; this will be reported elsewhere.

[14] http://numpy.scipy.org

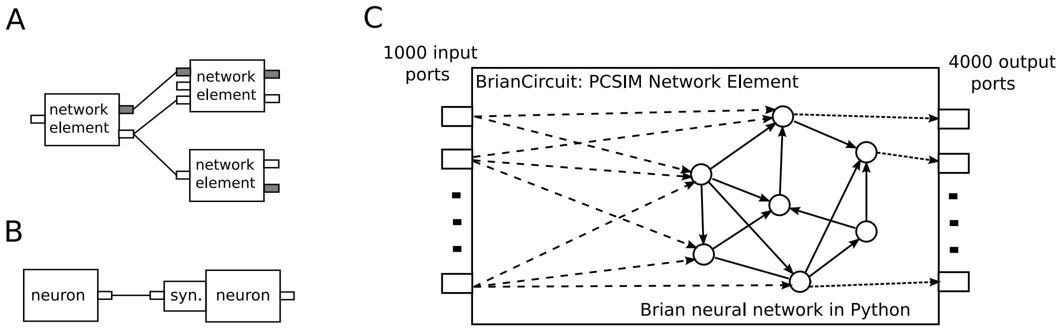

Figure 4.5: A) Network elements of different type (with different arrangement of input and output ports) interconnected together in a PCSIM network. Different colors of ports, gray or white, mark their different types, spiking or analog. B) Neurons and synapses are specific subtypes of the more general concept of an network element. C) Schematic diagram of the embedding of a network simulated with the Brian simulator into a PCSIM network element.

simulator, (Goodman and Brette, 2008)).

We detail such an example in this section, where the Brian simulator is used to implement a population of spiking neurons as a *single* network element, and then plug it into a PCSIM simulation together with other built-in network elements.

The spiking neural network model we will simulate with Brian is the modified version of the CUBA benchmark model described in Sec. 4.2.2, with a network size of 4000 neurons. We have used the same connectivity probability of 0.02 and the same weights as in (Brette et al., 2007), instead of the modified 0.1 connectivity probability and scaled weights in Sec. 4.2.2. The PCSIM network element that we will create to encapsulate the Brain network has 1000 spiking input ports and 4000 spiking output ports (see Fig. 4.5C). Each of the output ports is associated to one neuron.

To implement this model as a PCSIM network element, one has to implement a Python class `BrianCircuit` derived from `PySimObject`. In the constructor of this class the Brian spiking network is created and initialized.

```
class BrianCircuit(PySimObject):

    def __init__( self ):
        PySimObject.__init__( self )
        self.registerSpikingOutputPorts(arange(4000))
        self.registerSpikingInputPorts(arange(1000))
        input = PCSIMInputNeuronGroup(1000, self)
        self.P = P = brian.NeuronGroup(4000, model = eqs,
                                       threshold=-50*mV, reset=-60*mV)
        Pe = P.subgroup(3200)
        Pi = P.subgroup(800)
        Ce = brian.Connection(Pe, P, 'ge' )
        Ci = brian.Connection(Pi, P, 'gi' )
        Ce.connect_random( Pe, P, p = 0.02, weight = 1.62*mV )
        Ci.connect_random( Pi, P, p = 0.02, weight = -9*mV )
        Cinp = brian.Connection( input, P, 'ge' )
        Cinp.connect_random( input, P, p = 0.1, weight = 3.5*mV)
        self.brian = brian.Network(input, P, Ce, Ci, Cinp )
        self.brian.prepare()
        self.brian.clock.set_duration(2.0*second)
```

The mapping of the PCSIM input ports to a Brian neuron group is managed by the simple auxiliary neuron group named `PCSIMInputNeuronGroup` (see the supplementary material for the implementation). The `reset` method resets the state of the network to time step $t = 0$, which is achieved by calling the `reinit` method of the Brian network, and initializing the membrane potential vector `P.V` to random values from an uniform distribution.

```
    def reset(self, dt):
        self.brian.reinit()
        self.P.V = -60*mV+10*mV*rand(len(self.P))
        return 0
```

The step-by-step iteration of the network is done in the overridden `advance` method which performs one time-step update of the Brian network with the `update` method and the `tick` method of the associated Brian clock object. At the end of each time step the generated spikes of the population are gathered and delivered to the output ports of the PCSIM network element.

```
    def advance(self, ai):
        self.brian.update()
        self.brian.clock.tick()
        self.setOutputSpikes( ai, self.P.get_spikes() )
        self.clearSpikeBuf()
        return 0
```

Note that no Python loops are present, the `setOutputSpikes` method that transfers the spikes is implemented in C++ in the base class `PySimObject`, so there is no performance loss caused by the transfer of spikes from Brian to PCSIM and vice versa.

The new `BrianCircuit` network element class can then be instantiated and added to a PCSIM simulation. The following code segment creates an instance of the Brian spiking network, adds it as a network element, sets up the input and runs the simulation for 2.0 seconds (1000 neurons that emit Poisson spike trains at rate 5 Hz (`PoissonInputNeuron`) are connected to the 1000 input ports of the Brian network element[15]).

```
net = SingleThreadNetwork()
inpNrnPop = SimObjectPopulation( net,
              PoissonInputNeuron( rate = 5, duration = 1000 ), 1000 )

pycirc    = BrianCircuit()
pycirc_id = net.add(pycirc)

for i in range(inpNrnPop.size()):
    net.connect(inpNrnPop[i], 0, pycirc_id, i)

net.reset()
net.simulate( 2.0 )
```

4.6 Extending PCSIM using C++

The object-oriented framework of PCSIM can be extended by the user at many different levels. Typical extensions of PCSIM include either implementations of new neuron and synapse types, or implementations of classes encapsulating custom construction rules in the network construction interface, as we have illustrated in the previous sections. By utilizing the features of the Boost.Python library and Py++, the extensions can be implemented either in pure Python as already shown or in C++.

For creating C++ extensions, PCSIM provides a tool that compiles the custom C++ classes, *automatically* generates the Python wrapper interface for these and packs everything into a separate Python extension module. In order to simplify the procedure of creating a custom extension, the user starts the implementation from an extension template contained in the PCSIM distribution. Let us assume that we want to implement two classes: a new neuron type `MyNeuron` and a new synapse type `MySynapse`. Once the C++ implementation is finished, there are three additional steps that have to be done to produce the PCSIM extension module.

First, the C++ source files of the extension have to be enlisted in the file `module_recipe.cmake`. This file is read by PCSIM's C++ build tool CMake[16].

[15]The `net.connect(src_id, src_port, dest_id, dest_port)` method connects the port number `src_port` of the element with id `src_id`, to the port number `dest_port` of the element with id `dest_id`.

[16]http://www.cmake.org

```
SET( MODULE_SOURCES
   src/MySynapse.cpp
   src/MyNeuron.cpp
)
```

As the second step, we have to specify the names of the classes we want to include in the Python interface in the file `python_interface_specification.py` which holds the extension module interface specification. For our example the inserted lines should look like:

```
def specify( M, options ):
    M.class_( 'MySynapse' ).include()
    M.class_( 'MyNeuron' ).include()
    return M
```

Note that the argument `M` in the code above represents the Py++ representation of the C++ code of the custom PCSIM extension to be built, with its rather intuitive query interface.

The name of the extension module (in our example `my_pcsim_module`) is specified in both `module_recipe.cmake` and `python_interface_specification.py` files. Finally, the compilation is done using the special purpose command-line compilation tool for PCSIM extensions:

```
> Python pcsim_extension.py build
```

The compiled extension module then can be imported and used within Python as any other module.

```
import pypcsim
import my_pcsim_module
```

The main pypcsim module should always be imported before any PCSIM extension modules, because the classes in the extension are derived from classes in pypcsim and these classes should be already in the Python namespace. The user can develop multiple PCSIM extension modules that can be used simultaneously in one simulation.

The creation of PCSIM extensions as a separate Python extension module relies on the support of Boost.Python and Py++ for component-based development, so that C++ types from one Python extension module can be passed to functions from another extension module while still preserving the information about the cross-module C++ inheritance relationships. This enables object instances from the classes in the extension module to be used within the PCSIM object-oriented framework in the main pypcsim module. The component-based development has also the advantage that during the development of new custom classes only the extension module has to be recompiled, not the whole pypcsim library.

During the compilation of the PCSIM extension module the same processing steps happen as for the main pypcsim module (see Fig. 4.3). We use the same scripts both for generation of the Python interface of the main PCSIM package and for the Python integration of PCSIM extension modules. Since the post-processing exclusion rules are expressed with the Py++ query interface in a generic way, they are applicable also to the wrapping of the extension classes. This is due to the fact that extension classes are derived from base classes in the PCSIM object-oriented framework and as such share their common properties on which the rules are based. Hence, the interaction of the user with the interface generation and the module compilation reduces to specifying a list of the C++ source files, and a list of classes to be exposed in Python. The rest of the process is automatized and the details are hidden behind the command-line interface of the special compilation tool for PCSIM extensions.

4.7 PCSIM add-ons implemented in Python

On top of the main PCSIM Python API (encapsulated in pypcsim) several additional packages have been developed. They are implemented in pure Python and heavily rely on many third party scientific Python packages. The purpose of these packages is either to augment the capabilities of PCSIM, or add additional separate functionalities that are suitable to be used together with PCSIM.

4.7.1 PyNN.pcsim

The objective of the PCSIM development to adopt ongoing initiatives to define standards for model specification of neural networks that would foster interoperability between different simulators is reflected in the support of the PyNN project[17] (Davison et al., 2008). The PyNN project is an effort to create a standardized, unified Python-based API for procedural specification of neural network models aiming at easier exchange of models between simulators. The user interface of PCSIM has been augmented with an additional software layer to support the PyNN API making it possible to use models specified in PyNN within PCSIM. Due to the fact that PyNN was one of the sources for inspiration of the PCSIM interface, the concepts between the two interfaces match closely, so the translation of the PyNN statements in corresponding PCSIM statements was straightforward and did not require substantial programming logic that could have hindered the performance of the interface. The `pyNN.pcsim` package is an integral part of the PyNN distribution.

4.7.2 pypcsimplus

After we started to use PCSIM for our simulation purposes, it was becoming apparent that adding another layer above the interface of the pypcsim module can greatly simplify the routine tasks that are usually performed while setting up and running simulations. The pypcsimplus package was created with the intention to fill this gap. Note that the pypcsimplus package is dependent on PCSIM. For a more

[17] http://neuralensemble.org/trac/PyNN/

comprehensive, simulator independent tool-set for neural simulations, we refer the reader to the NeuroTools package[18]. In the following paragraphs we will describe two main components of the pypcsimplus package and give a demonstration of its use[19].

Recordings. In PCSIM the value of a parameter or output port is recorded during a simulation by connecting it to a proper recording network element. The purpose of the `Recordings` class is to provide simpler means to set up recorders and saving the recorded data during a PCSIM simulation. For example it allows to create a population of recorders that record the activity of a population of elements with each recorder connected to one of the elements (e.g. the spiking output of a population of neurons). For example

```
r = Recordings(net)

r.spikes  = nrn_popul.record( SpikeTimeRecorder() )
r.Vm      = net.record( my_nrn, ‘‘Vm’’, AnalogRecorder() )
r.weights = synapses.record( AnalogRecorder( samplingTime ), ‘‘W’’ )
```

schedules the recording of all spikes in the population `nrn_popul`, the membrane potential `Vm` of a single neuron (`my_nrn`), and the weights of a group of plastic synapses. To save that data to an HDF5 file[20] one would use the command

```
r.saveInOneH5File(f)
```

At any time later on, the saved data can be loaded from the file in a new `Recordings` object.

```
r = constructRecordingsFromH5File(f)
plot(r.Vm)
```

The members and attributes of the newly created `Recordings` object `r` are `numpy` arrays or Python lists holding the recorded data. For example `r.Vm` and `r.W` will be `numpy` arrays with the recorded values of the membrane potential of the neuron and with the evolution of the recorded synaptic weights during the simulation, respectively. Note that if the user switches to a distributed simulation the same code, without any changes, can be used.

To summarize, the `Recordings` class simplifies the specification, storage and retrieval of recorded data by

- providing automatic detection of the type of the recorded data based on the recorder classes, and conversion of the recorded data to appropriate HDF5

[18]http://neuralensemble.org/trac/NeuroTools

[19]There are other miscellaneous utilities present within the pypcsimplus package, as for example tools for easier management of IPython parallel computing cluster instances, routines for inspection of the structure of an already created networks in PCSIM and routines for processing and analysis of spike train data.

[20]http://www.hdfgroup.org/HDF5/

data structures.

- implementing automatic gathering and sorting of recorded data from all processing nodes in a distributed simulation, and saving it in HDF5 in the same format as if the simulation was executed on a single node.

These functionalities are hidden behind a convenient user interface and are manipulated in the same manner in both single-node and distributed simulation modes. For the implementation of the `Recordings` class, the mpi4py[21] (Dalcín et al., 2008) and pytables[22] packages were used.

Experiment-Model Framework. Simulation, modeling and development environments in various fields (e.g. electronic circuit design, software engineering, signal processing, mechanical engineering) usually include a library of already developed reusable components that are readily available to the modeler. In the area of computational neuroscience, there is a similar effort to provide resources for easier reusability of models, e.g. online databases of already published models (Hines et al., 2004), or constructs within the simulator that allow encapsulation of a simpler model as a well-defined component that can be used as a building block at a higher-level of abstraction. As a first step towards a component-based modeling with PCSIM, we have set up a light-weight framework that could leverage and encourage encapsulation of some generic parts of a model as reusable components, which can be exchanged among modelers.

The basis of the framework is composed of three classes: `Model`, `Experiment` and `Parameters`. The `Model` is a base class which the user inherits from when he wants to develop a model component. Several model components can be combined together to create a new model component. The `Experiment` class provides means to perform a controlled simulation with an already developed custom `Model` class. It encapsulates different facilities regarding saving output data to files, configuration of models, saving the current version of the scripts, naming of different runs of experiments etc. The configuration of the models is done with a `Parameters` class holding the model parameters in a hierarchical structure. For creating instances of the `Experiment` and `Model` classes remotely within the IPython parallel computing framework[23] (Pérez and Granger, 2007) there are `RemoteExperiment` and `RemoteModel` proxy classes, which can be used to manipulate remote experiment and model instances in the same way as if they were local.

pypcsimplus in action. We will demonstrate in the following paragraphs how pypcsimplus, together with other general scientific and computational neuroscience Python packages, can be utilized to perform an analysis of the activity of the Brian spiking network example from Sec. 4.5. In particular we will investigate what effect

[21] http://mpi4py.scipy.org
[22] http://www.pytables.org/moin
[23] http://ipython.scipy.org

a change in the injected input in the network will have on the cross-correlogram of its spike response.

At the beginning we will set up the recording of the spiking output of all 4000 neurons in the network. After creating a `Recordings` object, we create a population of recorders to record the spikes from the 4000 output ports of the `BrianCircuit` network element.

```
r = Recordings()
r.spikes = record_ports(net, pycirc_id, range(4000),
                                         SpikeTimeRecorder() )

net.simulate(2.0)

r.saveInOneH5File('results.h5')
```

We have accomplished this by using the `record_ports` function from the `pypcsimplus` package, used to specify recording of a set of output ports. After the simulation is performed, the recordings are saved in a HDF5 file for subsequent retrieval.

In another script we setup the analysis of the output data and the plotting. After the creation of the `Recordings` object by loading the recorded data from the saved HDF5 file, we plot the spiking activity of the network for the first 0.4 seconds of the simulation with the `plot_raster` function in `pypcsimplus` (see Fig. 4.6A).

```
r = constructRecordingsFromH5File('results.h5')

figure(1)
plot_raster(r.spikes, time_range = (0,0.4), fmt = ',')
```

`plot_raster` uses the plotting routines from the `matplotlib`[24] package (Hunter, 2007) to realize the plotting.

Additionally we will calculate and plot the cross-correlogram of the spiking activity, defined as the histogram of time differences between the spike times from two different spike trains, calculated and summed over a set of randomly chosen pairs of neurons from the network. To achieve this, we utilize the `pypcsimplus` function `avg_cross_correlate_spikes`.

```
corr = avg_cross_correlate_spikes(r.spikes, num_pairs = 2000,
                         binsize = 1e-3, corr_range = (-200e-3,200e-3) )

figure(2)
bar(arange(-200e-3,201e-3, 1e-3), corr, width = 1e-3, color = 'k')
```

In our case the cross-correlogram is calculated from the spike times of 2000 randomly chosen pairs of neurons from the network, for time lags within the range

[24]http://matplotlib.sourceforge.net

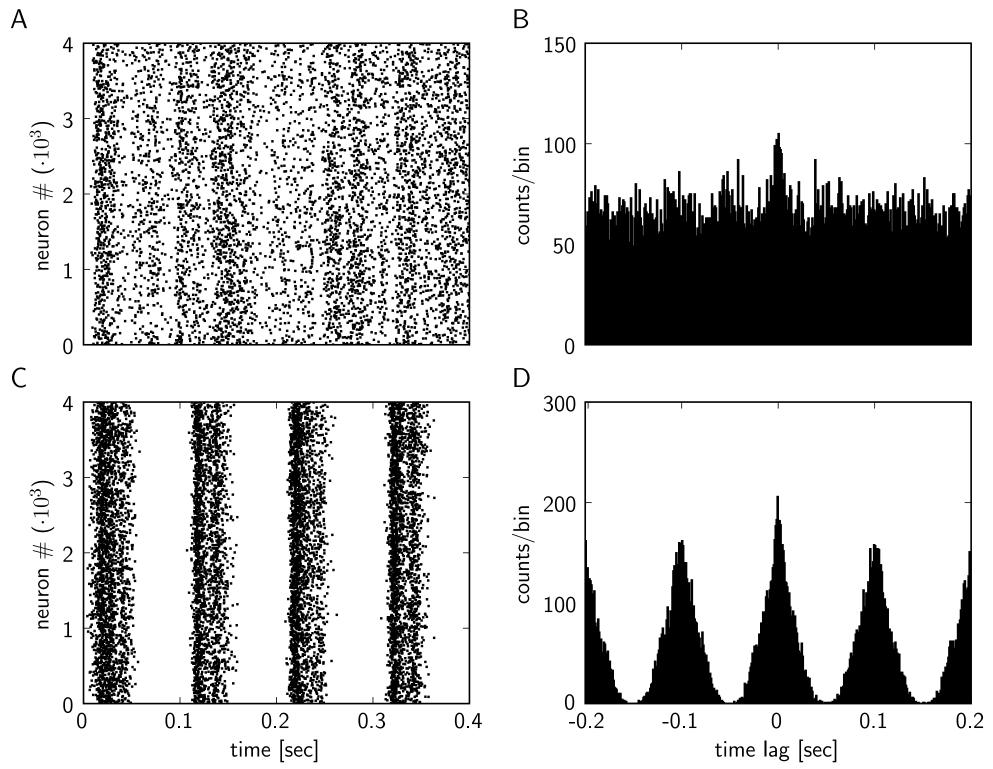

Figure 4.6: Plots from the output analysis example with the `pypcsimplus` package. A) Spike response of the spiking network implemented in Sec. 4.5, with input neurons emitting spikes generated from a homogeneous Poisson process with a rate of 5 Hz, for the first 0.4 seconds of the simulation. B) Cross-correlogram of the spike response of the network model from A). C) Spike response of the spiking network implemented in Sec. 4.5, when the input neurons emit spikes generated from a inhomogeneous Poisson process with a rate changing according to a sinusoidal function (see text for details). D) Cross-correlogram of the spike response of the network model from C).

$[-200ms, 200ms]$ and a bin size of 1ms. We then plot the cross-correlogram values with the `bar` function from `matplotlib` (the plot is shown in Fig. 4.6B)[25]

In the example in Sec. 4.5, the input neurons were setup to generate a homogeneous Poisson spike trains with 5 Hz rate. Now we will modify the input generation so that the input neurons will emit inhomogeneous Poisson spike trains, with a firing rate $r(t) = 5(1 + \sin(2\pi 10t))$. First we create a population of input neurons of type `SpikingInputNeuron` that emit an explicitly given sequence of spike times.

```
inpNrnPop = SimObjectPopulation(net, SpikingInputNeuron(), 1000)
```

Then we iterate through all the input neurons and set the spike sequence of each input neuron according to the previously defined inhomogeneous Poisson process.

[25]For clarity reasons, we only give the main matplotlib plotting command in the example code blocks, and omit the additional formatting commands used for Fig. 4.6.

For the generation of the inhomogeneous Poisson spike time sequences we invoke the `inh_poisson_generator` method of an instance of the `StGen` (stimulus generator) class available in the `NeuroTools` Python package for computational neuroscience. The method accepts three parameters, a sequence specifying the time moments where the rate changes (parameter `t`), the sequence of the new firing rate values at these time moments (parameter `rate`) and the duration of the spiking process (parameter `t_stop`)[26].

```
time_steps = arange(0,2000,1); stgen = StGen()
for i in range(inpNrnPop.size()):
        spikelist = stgen.inh_poisson_generator(
                        rate = 5*(1 + sin(time_steps/1000.0*20*pi)),
                        t = time_steps, t_stop = 2000.0)
        inpNrnPop.object(i).setSpikes(spikelist.spike_times/1000)
```

The spike raster and the cross-correlogram obtained after rerunning the simulation with the newly defined input are shown in Fig. 4.6, panels C and D, respectively.

Through this demo we have elucidated to the reader how a typical PCSIM simulation run is performed in Python, and the benefits that come from the utilization of Python as a unifying scripting environment within which PCSIM is used together with its add-on `pypcsimplus` and other scientific and computational neuroscience Python packages. Additionally to their side-by-side usage with PCSIM, the Python scientific packages are harnessed also in the bundling of common recipes and reoccurring usage patterns in the PCSIM extra add-on packages, as in the case of pypcsimplus. The collection of Python scientific packages presently available cover a broad enough range of functionalities to enable, in almost all cases, handling all of the steps of a modeling effort in Python (e.g. stimulus preparation, response analysis and plotting as shown in the demo). The data communication between the different packages and PCSIM typically reduces to passing Python sequences (lists or `numpy` arrays) from one package to another.

4.7.3 pylsm

The pylsm package is aimed to support the analysis of the computational properties of cortical microcircuits within the liquid state machine (LSM) approach (Maass et al. (2002c)). In this approach multiple simulation trials are performed where input spike trains, drawn from a defined input distribution, are injected in the cortical circuit, and a readout which reads the spiking activity of the circuit is trained by a supervised learning algorithm to approximate some function of these inputs.

The framework contains all the necessary machinery for performing the simulations and the training of the readout[27]. In a typical task the user defines the neural

[26]Time in neurotoools is specified in milliseconds, hence the division by 1000 when we need to convert the spike time sequence in seconds before inserting it in a PCSIM neuron.

[27]It has similar features as the package described in (Natschläger et al., 2003), which was imple-

circuit to be used as a liquid, chooses the desired input distribution, the input-output mapping function, and the learning algorithm for the readout from the ones available in the package, and then performs the LSM training and testing procedures. For example, the user can define a distribution of inputs which consist of different time segments, and each of these time segments contains a jittered version of some predefined spike train template. In the available learning algorithms for the readout a least-square algorithm with non-negative constraints is also included. It can be used to train a linear readout with the biologically more realistic constraint that all the weights originating from excitatory (inhibitory) neurons are positive (negative) (Haeusler and Maass, 2007).

4.8 Discussion

The application programming interface of PCSIM is an object-oriented framework composed of many classes interacting together to achieve the desired operation. Within this framework we introduced several novel concepts like element and connector factories, value generators and connection decision predicates. The user can customize and extend this framework by deriving from the interface classes of the API to implement his own specific network elements or network construction algorithms.

The wrapping approach. There exist several possible approaches for implementing a Python interface of a simulation software library implemented in C/C++. An extension to the NCS software called Brainlab (Drewes, 2005) uses generation of a file from Python with declarative specification of the model which is then loaded in the simulator. Another common method is to use interpreter-to-interpreter interaction with the conversion of data structures between Python and C++ handled by means of the Python/C API, an approach adopted by NEURON (Hines et al., 2009) and NEST (Eppler et al., 2008). This method is applicable only if the simulator already has an interpreting interface. For the creation of PyMoose (Ray and Bhalla, 2008), the Python interface of MOOSE (http://moose.sourceforge.net/), the developers applied the interface generator tool SWIG[28] (Beazley, 2003). Certainly, one can also implement a Python interface by using solely the Python/C API.

Since PCSIM's interface was to be newly developed, only the later two options were applicable. We opted for the interface generator tool approach combined with automatic wrapper code generation, since from the available options it seemed to us the fastest way, in terms of the amount of development effort required, to achieve the desired Python wrapping of the PCSIM object-oriented framework. One of our goals for the integration of PCSIM with Python was to simplify and support a hybrid modeling approach by enabling the user to implement extensions of the PCSIM object-oriented framework in Python and/or C++, while not having to

mented in Matlab and was part of the CSIM package.

[28] http://www.swig.org

bother with details regarding the interoperability between these two programming languages.

The excellent support of Boost.Python for advanced C++ concepts and appropriate mapping of corresponding idioms between the two languages allowed us to expose the complete PCSIM API, currently $\approx$ 300 classes, to Python in a non-intrusive way. This means that the fact that the PCSIM API is to be exposed to Python does not impose any changes at the C++ level nor does it put any constraints on its design. Furthermore the compilation of the libpcsim library itself does not depend on any Python library or wrapping code.

Bidirectional interface and hybrid model definition. One of the features of Boost.Python enabling the hybrid approach is the ability to derive Python classes from the wrapped interface classes, and override the virtual functions. Hence, such custom Python class methods can be called from within C++ and thus allow an integration of Python code into the PCSIM C++ code. A similar bidirectional interface has been implemented between Python and NEURON (Hines et al., 2009), where Python can issue commands towards NEURON, but also Python code can be called and executed from within NEURON in an active Hoc session [29]. In PCSIM the two-way interaction between Python and C++ enables user customizations to be coded in pure Python, and then plugged into the PCSIM C++ framework. This brings additional flexibility and freedom to the user, meaning that he can first do fast implementations in Python, e.g. extensions to the network construction interface (Sec. 4.4), in the prototyping phase, and afterwards the implementation can be ported to C++ to gain maximum performance.

The ability to define PCSIM network elements in Python opens a possibility for a seamless Python-C++ integration also during the simulation, not only in the network construction stage. The example described in Sec. 4.5 shows that network elements can be implemented in Python, by using vectorized techniques employing the highly efficient numerical Python package numpy (which is implemented in C). This adds flexibility, since the equations describing the element can be changed quickly without any necessary compilation while not sacrificing performance, since by using numpy vectors, the integration algorithm is broken down in elementary vector operations thus avoiding any loops within Python that could be detrimental for the performance.

This approach seems also to be advantageous when one wants to implement network elements that have some abstract processing logic, e.g. signal processing filters, machine learning algorithms or similar. In this case one can utilize a large set of available C++ libraries that have Python bindings, for an efficient implementation, and handle in Python the transfer of data from the input ports of the network element to the input methods of the library, and from the output of the library to the output ports of the network element.

The possibility to implement PCSIM network elements in pure Python offers

[29]Hoc is the native NEURON interpreting language.

a convenient way to achieve run-time interoperability between PCSIM and other neural network simulators (Cannon et al., 2007), provided that the simulator has a Python interface, allows control of the simulation process at individual time steps, and has the possibility to write input and read output data during the simulation at each time step. As shown in the example in Sec.4.5, we have successfully implemented interoperability with the Brian simulator, which possesses the aforementioned capabilities. One interesting further application of this interoperability could be a distributed simulation of a large neural network where the sub-networks on each node are implemented with the Brian simulator, and the parallel communication is handled by PCSIM's communication system. Another possible approach of using Python as a glue language to achieve simulator interoperability is to setup a Python script as a top-level coordinator of a step-by-step simultaneous execution of two simulators, where the necessary data transfer between the simulators is realized through intermediate Python data structures (Ray and Bhalla, 2008).

High-level wrapping specification and extensibility. Since the interface has a fine granular structure, composed of many decoupled classes (≈ 300) this implies that there are many classes to be wrapped and exposed to Python. It would simply be impossible to manually manage all the necessary Boost.Python wrapper code. Furthermore, the possibility of adding extensions to the interface puts additional constraints to the wrapping approach to be robust enough to work for the extension classes too, without any significant intervention from the user. Nevertheless, by exploiting the powerful interface generator tool Py++ the wrapping of such a large number of classes is rendered feasible.[30] We were able to specify high-level generic rules within Py++ for the definition of the wrapping of all the classes in the PCSIM API and their sensible extensions. To be precise, the Python program that specifies the rules for the Python interface generation for ≈ 300 classes is about 400 lines of Python code. As these rules apply for the extensions too, the user can easily extend the PCSIM simulator with its own custom C++ classes and compile them in a separate Python extension package, which can be used together with the main `pypcsim` package (the tool support for this is included in PCSIM). This was made possible by the Boost.Python and Py++ support for cross-module inheritance relationships and component-based development (see Sec. 4.6).

To summarize, by the easy extensibility of its interface both in Python and C++, PCSIM enables the modelers to *think hybrid* when developing their models (Abrahams and Grosse-Kunstleve, 2003).

Python as a scripting environment. Providing a Python interface to a neural simulator increases its versatility and consequently the productivity of the modelers in many ways. The object oriented design of the language, its expressive and clean syntax, allows the modeler to focus on the high-level logic of the model in-

[30]The only drawback we encounter is the rather long compile time when recompiling the whole Python interface. This is due to the fact that Boost.Python heavily uses C++ templates.

stead of struggling with the intricacies and the nuts and bolts of the programming language. Furthermore, there is a growing number of general scientific and specific computational neuroscience software tools available as Python packages, for numerical calculations, scientific functions, plotting, saving data to files, parallel computing etc. We have used several scientific Python packages to enhance PCSIM with useful utilities on top of its basic interface. As we have illustrated through a simple example in Sec. 4.7, in combination with such Python packages PCSIM can be used as the main component of a Python-based neural simulation environment where all steps within a neural model development life-cycle, from the specification of the model and performing the simulations, to storage of simulation output data, data analysis and visualization can be performed. Overall, the integration of PCSIM with Python added additional valuable facilities to the user, turning PCSIM into a full-fledged neural simulation environment.

PCSIM Resources. Many resources for PCSIM can be found at its web page `http://www.igi.tugraz.at/pcsim`. The web page contains a user manual, examples, installation instructions, complete class reference documentation and the complete material for the tutorial that was given at the FIAS Theoretical Neuroscience and Complex Systems summer school held in Frankfurt, Germany in August, 2008. The users can discuss topics and pose questions concerning usage and installation of PCSIM on the *pcsim-users* mailing list on Sourceforge®(`http://www.sourceforge.net/projects/pcsim`) where the PCSIM development project is hosted. In the future, the user manual will continuously undergo extensions and revisions to better organize the content and to include additional topics and more elaborate information about the PCSIM concepts and constructs. Additional examples covering various PCSIM features will also be made available on the web site.

4.9 Acknowledgments

This chapter is based on the journal article *PCSIM: A Parallel Simulation Environment for Neural Circuits Fully Integrated with Python* by myself (DP), Thomas Natschläger (TN) and Klaus Schuch (KS) (*Frontiers in Neuroinformatics* 3:11, 2009). The PCSIM neural simulator described in the article was developed by DP and TN, with contributions from KS. TN supervised the software development project. DP implemented and performed the computer simulation tests reported in the article. The article was written by DP and TN. KS wrote the section that describes the PYLSM package and gave useful comments for improving the manuscript.

Appendix A

List of Publications

1. R. Brette, M. Rudolph ,T. Carnevale,M. Hines,D. Beeman,J.M. Bower, M. Diesmann, A. Morrison, P.H. Goodman, F.C. Harris Jr.,M. Zirpe , T. Natschläger, D. Pecevski, B. Ermentrout , M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A.P. Davison, S. El Boustani, and A. Destexhe. *Simulation of networks of spiking neurons: a review of tools and strategies*, Journal of Computational Neuroscience 23(3):349-398, 2007.

2. R. Legenstein[1], D. Pecevski[1], and W. Maass *Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity*, In Proc. of NIPS 2007, Advances in Neural Information Processing Systems, volume 20. MIT Press, 2008.

3. Legenstein R., Pecevski D. and Maass W. *A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback*, PLoS Computational Biology 4(10): e1000180, 2008.

4. A.P. Davison , D. Brüderle, J. Kremkow , E. Muller , Pecevski D., Perrinet, L. and P. Yger, *PyNN: a common interface for neuronal network simulators*, Frontiers in Neuroinformatics. Conference Abstract: Neuroinformatics 2008.

5. A.P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet and P. Yger, *PyNN: a common interface for neuronal network simulators*, Frontiers in Neuroinformatics 2:11, 2008.

6. D. Pecevski, T. Natschläger and K. Schuch *PCSIM: A Parallel Simulation Environment for Neural Circuits Fully Integrated with Python*, Frontiers in Neuroinformatics 3:11, 2009.

7. E. Muller, A. P. Davison, T. Brizzi, D. Bruederle, M. J. Eppler, J. Kremkow, D. Pecevski, L. Perrinet, M. Schmuker and P. Yger (2009) *NeuralEnsemble.Org: Unifying neural simulators in Python to ease the model complexity bottleneck*, Frontiers in Autonomic Neuroscience. Conference Abstract: Neuroinformatics 2009.

8. D. Pecevski, L. Büsing, W. Maass. *Probabilistic Inference in General Graphical Models through Sampling in Stochastic Networks of Spiking Neurons*, submitted for publication, 2011.

[1]These authors contributed equally to the paper

A.1 Comments and Contributions to Publications

The first publication *Simulation of networks of spiking neurons: a review of tools and strategies* is a review publication which overviews different simulation environments for networks of spiking neurons. In this publication I prepared and performed the benchmark simulations for the simulators CSIM and its successor PCSIM.

The publication *Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity* was written by Robert Legenstein (RK), myself (DP) and my supervisor Wolfgang Maass (WM). RK provided the theoretical analysis, RL, DP and WM concieved the experiments and DP prepared and performed the simulations for the experiments and analysed the simulation results. RL, DP and WM wrote the paper. The paper was selected for a *spotlight poster* presentation at the *21th Annual Conference on Neural Information Processing Systems* (NIPS) 2007, Vancouver, Canada. The results from this paper were extended and published in a journal article *A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback* by the same authors, published in *PLOS Computational Biology*. Apart from containing in a more elaborate form the results from the conference publication, the journal publication also includes additional theoretical analysis and additional results from elaborate simulation experiments. In this article RL contributed the theoretical analysis, RL, DP and WM concieved the experiments and DP conducted the simulation experiments and analysed the simulation results. RL, DP and WM wrote the paper. The journal article provides the basis for Chapter 3 of this thesis.

The journal publication *PyNN: a common interface for neuronal network simulators* published in *Frontiers in Neuroinformatics* describes the software package PyNN, a simulator-independent Python-based interface for specification and simulation of models composed of networks of spiking neurons. All those who contributed code to PyNN were added as co-authors of this article. I contributed to PyNN the module that implements the support for the PCSIM simulator.

The journal article *PCSIM: A Parallel Simulation Environment for Neural Circuits Fully Integrated with Python* published in *Frontiers in Neuroinformatics* gives an overview of the functionalities of the PCSIM simulator and its integration with the Python programming language. The PCSIM simulator was developed by myself (DP) and Thomas Natschläger (TN), with contributions from Klaus Schuch (KS). DP implemented and performed the computer simulation tests reported in the article. The paper was written by DP and TN. KS wrote the section that describes the PYLSM package. This article provides the basis for Chapter 4 of this thesis.

The article *Probabilistic Inference in General Graphical Models through Sampling in Stochastic Networks of Spiking Neurons* is a joint work together with Lars Büsing (LB) and Wolfgang Maass (WM). It was submitted for publication in 2011 and is under review. The experiments were concieved and designed by myself (DP) and WM. DP conducted the experiments and analysed the simulation results. The paper builds on the theory of neural sampling developed by LB and reported in (Büsing et al., 2011). DP and WM provided the additional theoretical derivations and analysis in the paper. DP and WM wrote the paper. LB provided valuable comments that helped to improve the paper. This article provides the basis for Chapter 2 of this thesis.

Bibliography

Abbott, L. F. and Nelson, S. B. (2000). Synaptic plasticity: taming the beast. *Nature Neuroscience*, 3:1178–1183. 56, 58

Abeles, M., Bergman, H., Gat, I., Meilijson, I., Seidemann, E., Tishby, N., and Vaadia, E. (1995). Cortical activity flips among quasi-stationary states. *Proc Natl Acad Sci U S A*, 92(19):8616–8620. 30, 37

Abrahams, D. and Grosse-Kunstleve, R. W. (2003). Building hybrid systems with Boost.Python. *C/C++ Users Journal*, 21(7):29–36. 109, 113, 134

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive Science*, 9:147–169. 9, 16, 34

Anderson, J., Lampl, I., Reichova, I., Carandini, M., and Ferster, D. (2000). Stimulus dependence of two-state fluctuations of membrane potential in cat visual cortex. *Nature Neuroscience*, 3(6):617–621. 93

Andrieu, C., Freitas, N. D., Doucet, A., and Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, 50:5–43. 9

Bailey, C. H., Giustetto, M., Huang, Y.-Y., Hawkins, R. D., and Kandel, E. R. (2000). Is heterosynaptic modulation essential for stabilizing Hebbian plasticity and memory? *Nature Reviews Neuroscience*, 1:11–20. 56

Bao, S., Chan, V. T., and Merzenich, M. M. (2001). Cortical remodelling induced by activity of ventral tegmental dopamine neurons. *Nature*, 412(6842):79–83. 56

Baras, D. and Meir, R. (2007). Reinforcement learning, spike-time-dependent plasticity, and the bcm rule. *Neural Computation*, 19(8):2245–2279. 56, 103

Baxter, J. and Bartlett, P. L. (1999). Direct gradient-based reinforcement learning: I. gradient estimation algorithms. Technical report, Research School of Information Sciences and Engineering, Australian National University. 56, 103

Beazley, D. (2003). Automated scientific software scripting with SWIG. *Future Generation Computer Systems*, 19(5):599 – 609. 132

Beck, J. M. and Pouget, A. (2007). Exact inferences in a neural implementation of a hidden Markov model. *Neural Computation*, 19(5):1344–1361. 34

Berkes, P., Orban, G., Lengyel, M., and Fiser, J. (2011). Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment. *Science*, 331:83–87. 36

Bi, G. and Poo, M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J Neuroscience*, 18(24):10464–10472. 59

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing edition. 8

Bobrowski, O., Meir, R., and Eldar, Y. C. (2009). Bayesian filtering in spiking neural networks: Noise, adaptation, and multisensory integration. *Neural Computation*, 21(5):1277–1320. 34

Borg-Graham, L. J., Monier, C., and Frégnac, Y. (1998). Visual input evokes transient and strong shunting inhibition in visual cortical neurons. *Nature*, 393:369–373. 93

Bower, J. M. and Beeman, D. (1998). *The book of GENESIS (2nd ed.): exploring realistic neural models with the GEneral NEural SImulation System*. Springer-Verlag New York, Inc., New York, NY, USA. 108

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., Diesmann, M., Morrison, A., Goodman, P. H., Jr., F. C. H., Zirpe, M., Natschläger, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A. P., Boustani, S. E., and Destexhe, A. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398. 108, 111, 122

Brunel, N. (2000). Dynamics of networks of randomly connected excitatory and inhibitory spiking neurons. *Journal of Physiology-Paris*, 94:445–463. 58, 100

Büsing, L., Bill, J., Nessler, B., and Maass, W. (2011). Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons. *submitted for publication*. 4, 5, 10, 11, 12, 13, 15, 16, 17, 19, 32, 36, 38, 39, 42, 43, 50, 53, 138

Buzas, P., Kovacs, K., Ferecsko, A. S., Budd, J. M. L., Eysel, U. T., and Kisvarday, Z. F. (2006). Model-based analysis of excitatory lateral connections in the visual cortex. *J Comp Neurol*, 499(6):861–81. 115

Cannon, R., Gewaltig, M.-O., Gleeson, P., Bhalla, U., Cornelis, H., Hines, M., Howell, F., Muller, E., Stiles, J., Wils, S., and Schutter, E. D. (2007). Interoperability of neuroscience modeling software: Current status and future directions. *Neuroinformatics*, 5(2):127–138. 134

Carnevale, N. T. and Hines, M. L. (2006). *The NEURON Book*. Cambridge University Press, New York, NY, USA. 108

Churchland, P. S., Koch, C., and Sejnowski, T. J. (1993). *What is computational neuroscience?*, pages 46–55. MIT Press, Cambridge, MA, USA. 1

Dalcín, L., Paz, R., Storti, M., and D'Elía, J. (2008). Mpi for python: Performance improvements and mpi-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655–662. 128

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., and Yger, P. (2008). PyNN: a common interface for neuronal network simulators. *Front. Neuroinform.*, 2(11). 115, 126

Dean, A. F. (1981). The variability of discharge of simple cells in the cat striate cortex. *Experimental Brain Research*, 44:437–440. 36

Deneve, S. (2008). Bayesian spiking neurons I: Inference. *Neural Computation*, 20(1):91–117. 34, 35

Deneve, S., Latham, P. E., and Pouget, A. (2001). Efficient computation and cue integration with noisy population codes. *Nat Neurosci*, 4(8):826–831. 34

Denison, S., Bonawitz, E., Gopnik, A., and Griffiths, T. (2010). Preschoolers sample from probability distributions. In *Proc. of the 32nd Annual Conference of the Cognitive Science Society*. 36

Destexhe, A. and Marder, E. (2004). Plasticity in single neuron and circuit computations. *Nature*, 431:789–795. 77, 101

Destexhe, A., Rudolph, M., Fellous, J. M., and Sejnowski, T. J. (2001). Fluctuating synaptic conductances recreate in vivo-like activity in neocortical neurons. *Neuroscience*, 107(1):13–24. 65, 78, 92, 93, 95

Douglas, R. J. and Martin, K. A. (2004a). Neuronal circuits of the neocortex. *Annu Rev Neurosci*, 27:419–451. 2

Douglas, R. J. and Martin, K. A. (2004b). Neuronal circuits of the neocortex. *Annual Review of Neuroscience*, 27(1):419–451. 34

Doya, K., Ishii, S., Pouget, A., and Rao, R. P. N. (2007). *Bayesian Brain: Probabilistic Approaches to Neural Coding*. MIT-Press. 8

Drewes, R. (2005). Modeling the brain with NCS and brainlab. *Linux Journal*, 2005(134):2. 132

Eppler, J. M., Helias, M., Muller, E., Diesmann, M., and Gewaltig, M.-O. (2008). Pynest: a convenient interface to the nest simulator. *Front. Neuroinform*, 2(12). 132

Farries, M. A. and Fairhall, A. L. (2007). Reinforcement learning with modulated spike timing-dependent synaptic plasticity. *Journal of Neurophysiology*, 98:3648–3665. 75, 100, 104

Fetz, E. E. (1969). Operant conditioning of cortical unit activity. *Science*, 163(870):955–958. 57

Fetz, E. E. (2007). Volitional control of neural activity: implications for brain-computer interfaces. *J Physiol*, 579(3):571–579. 57, 65, 68

Fetz, E. E. and Baker, M. A. (1973). Operantly conditioned patterns of precentral unit activity and correlated responses in adjacent cells and contralateral muscles. *J Neurophysiol*, 36(2):179–204. 5, 57, 62, 63, 64, 65, 67, 68, 99, 103

Fetz, E. E. and Finocchio, D. V. (1975). Correlations between activity of motor cortex cells and arm muscels during operantly conditioned response patterns. *Exp. Brain Research*, 23(3):217–240. 57

Fiete, I. R. and Seung, H. S. (2006). Gradient learning in spiking neural networks by dynamic perturbation of conductances. *Physical Review Letters*, 97(4):048104–1 to 048104–4. 103

Fiser, J., Berkes, P., Orbán, G., and Lengyel, M. (2010). Statistically optimal perception and learning: from behavior to neural representations. *Trends in Cognitive Sciences*, 14(3):119 – 130. 8, 36

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, 6:1468–1502. 56, 103

Gershman, S. J., Vul., E., and Tenenbaum, J. (2009). Perceptual multistability as Markov chain Monte Carlo inference. *Advances in Neural Information Processing Systems*, 22:611–619. 36

Gerstner, W. and Kistler, W. M. (2002). *Spiking Neuron Models*. Cambridge University Press, Cambridge. 58, 59, 60, 61, 82

Gewaltig, M.-O. and Diesmann, M. (2007). NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430. 108

Goodman, D. and Brette, R. (2008). Brian: a simulator for spiking neural networks in python. *Front. Neuroinform.*, 2(5). 109, 122

Griffiths, T. L. and Tenenbaum, J. B. (2006). Optimal Predictions in Everyday Cognition. *Psychological Science*, 17(9):767–773. 36

Grimmett, G. R. and Stirzaker, D. R. (2001). *Probability and Random Processes*. Oxford University Press, 3rd edition. 38

Gu, Q. (2002). Neuromodulatory transmitter systems in the cortex and their role in cortical plasticity. *Neuroscience*, 111(4):815–835. 56

Gupta, A., Wang, Y., and Markram, H. (2000). Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex. *Science*, 287:273–278. 92

Gütig, R. and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nature Neuroscience*, 9(3):420–428. 103, 104

Haeusler, S. and Maass, W. (2007). A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models. *Cereb Cortex*, 17(1):149–62. 132

Hammarlund, P. and Ekeberg, O. (1998). Large neural network simulations on multiple hardware platforms. *Journal of computational neuroscience*, 5(4):443–459. 108

Häusler, S. and Maass, W. (2007). A statistical analysis of information processing properties of lamina-specific cortical microcircuit models. *Cerebral Cortex*, 17(1):149–162. 101, 102

Hines, M., Davison, A. P., and Muller, E. (2009). Neuron and python. *Front. Neuroinform*, 3(1). 132, 133

Hines, M. L. and Carnevale, N. T. (1997). The neuron simulation environment. *Neural Computation*, 9(6):1179–1209. 108

Hines, M. L., Morse, T., Migliore, M., Carnevale, N. T., and Shepherd, G. M. (2004). ModelDB: A database to support computational neuroscience. *Journal of Computational Neuroscience*, 17(1):7–11. 128

Hinton, G. E. and Brown, A. D. (2000). Spiking Boltzmann machines. In *In Advances in Neural Information Processing Systems 12*, Cambridge, MA. MIT Press. 35

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1 of *Lecture Notes in Computer Science*. MIT Press, Cambridge, MA. 34, 35

Hirsch, J. A., Alonso, J. M., Reid, R. C., and Martinez, L. M. (1998). Synaptic integration in striate cortical simple cells. *J. Neurosci.*, 18(22):9517–9528. 93

Hopfield, J. J. and Brody, C. D. (2001). What is a moment? Transient synchrony as a collective mechanism for spatio-temporal integration. *Proc. Nat. Acad. Sci. USA*, 98(3):1282–1287. 101

Hoyer, P. O. and Hyvärinen, A. (2003). Interpreting neural response variability as Monte Carlo sampling of the posterior. In S. Becker, S. T. and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 277–284. MIT Press, Cambridge, MA. 36

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science and Engineering*, 9(3):90–95. 129

Ide, J. and Cozman, F. (2002). Random generation of Bayesian networks. In Bittencourt, G. and Ramalho, G., editors, *Advances in Artificial Intelligence*, volume

2507 of *Lecture Notes in Computer Science*, pages 366–376. Springer Berlin / Heidelberg. 29, 52

Izhikevich, E. (2004). Which model to use for cortical spiking neurons? *Neural Networks, IEEE Transactions on*, 15(5):1063–1070. 115, 117

Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, 17:2443–2452. 56, 75, 79, 100, 103, 104

Jacob, V., Brasier, D., Erchova, I., Feldman, D., and Shulz, D. E. (2007). Spike timing-dependent synaptic depression in the in vivo barrel cortex of the rat. *J Neuroscience*, 27(6):1271–84. 56

Jolivet, R., Rauch, A., Lüscher, H.-R., and Gerstner, W. (2006). Predicting spike timing of neocortical pyramidal neurons by simple threshold models. *Journal of Computational Neuroscience*, 21:35–49. 11

Kempter, R., Gerstner, W., and van Hemmen, J. L. (1999). Hebbian learning and spiking neurons. *Phys. Rev. E*, 59(4):4498–4514. 82

Kempter, R., Gerstner, W., and van Hemmen, J. L. (2001). Intrinsic stabilization of output rates by spike-based hebbian learning. *Neural Computation*, 13:2709–2741. 63

Kenet, T., Bibitchkov, D., Tsodyks, M., Grinvald, A., and Arieli, A. (2003). Spontaneously emerging cortical representations of visual attributes. *Nature*, 425(6961):954–956. 36

Kersten, D. and Yuille, A. (2003). Bayesian models of object perception. *Current Opinion in Neurobiology*, 13(2):150 – 158. 11

Knill, D. C. and Kersten, D. (1991). Apparent surface curvature affects lightness perception. *Nature*, 351:228–230. 12, 14, 33

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques (Adaptive Computation and Machine Learning)*. MIT Press. 8, 9, 12, 32

Koulakov, A. A., Hromadka, T., and Zador, A. M. (2009). Correlated connectivity and the distribution of firing rates in the neocortex. *The Journal of Neuroscience*, 29(12):3685–3694. 37

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224. 25, 26, 49

Legenstein, R. and Maass, W. (2011). Branch-specific plasticity enables self-organization of nonlinear computation in single neurons. *The Journal of Neuroscience*. in press. 20, 34, 39, 45

Levin, D. A., Peres, Y., and Wilmer, E. L. (2008). *Markov Chains and Mixing Times*. American Mathematical Society. 32

Li, C. T., Poo, M., and Dan, Y. (2009). Burst spiking of a single cortical neuron modifies global brain state. *Science*, 324:643–646. 37

Litvak, S. and Ullman, S. (2009). Cortical circuitry implementing graphical models. *Neural Computation*, 21(11):3010–3056. 34

Losonczy, A., Makara, J. K., and Magee, J. C. (2008). Compartmentalized dendritic plasticity and input feature storage in neurons. *Nature*, 452:436–441. 20, 39, 40, 45, 46

Lyon, R. (1982). A computational model of filtering, detection, and compression in the cochlea. In *Proceedings of IEEE International Conference on ICASSP*, pages 1282–1285. 98, 105

Ma, W. J., Beck, J. M., Latham, P. E., and Pouget, A. (2006). Bayesian inference with probabilistic population codes. *Nat Neurosci*, 9(11):1432–1438. 34

Ma, W. J., Beck, J. M., and Pouget, A. (2008). Spiking networks for Bayesian inference and choice. *Current Opinion in Neurobiology*, 18(2):217 – 222. Cognitive neuroscience. 34

Maass, W., Joshi, P., and Sontag, E. D. (2007). Computational aspects of feedback in neural circuits. *PLoS Computational Biology*, 3(1):e165, 1–20. 77, 101, 102

Maass, W. and Markram, H. (2002). Synapses as dynamic memory buffers. *Neural Networks*, 15:155–161. 92

Maass, W., Natschlaeger, T., and Markram, H. (2002a). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560. 34

Maass, W., Natschläger, T., and Markram, H. (2002b). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560. 77, 78, 101, 102

Maass, W., Natschlager, T., and Markram, H. (2002c). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Comp.*, 14(11):2531–2560. 131

Maass, W., Natschläger, T., and Markram, H. (2004). Fading memory and kernel properties of generic cortical microcircuit models. *Journal of Physiology – Paris*, 98(4–6):315–330. 77, 101

Mainen, Z. and Sejnowski, T. (1995). Reliability of spike timing in neocortical neurons. *Science*, 268:1503–1505. 76

Mansinghka, V. K., Kemp, C., Tenenbaum, J. B., and Griffiths, T. L. (2006). Structured priors for structure learning. In *In Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press. 26

Markram, H., Wang, Y., and Tsodyks, M. (1998). Differential signaling via the same axon of neocortical pyramidal neurons. *Proc. Nat. Acad. Sci. USA*, 95:5323–5328. 91, 92

Miller, P. and Katz, D. (2010). Stochastic transitions between neural states in taste processing and decision-making. *J. of Neurosc.*, 30(7):2559–2570. 30, 37

Morrison, A., Aertsen, A., and Diesmann, M. (2007). Spike-timing-dependent plasticity in balanced random networks. *Neural Computation*, 19:1437–1467. 59, 66, 94, 100

Morrison, A., Mehring, C., Geisel, T., Aertsen, A., and Diesmann, M. (2005). Advancing the Boundaries of High-Connectivity Network Simulation with Distributed Computing. *Neural Comp.*, 17(8):1776–1801. 108, 110, 112

Natschläger, T., Markram, H., and Maass, W. (2003). Computer models and analysis tools for neural microcircuits. In Kötter, R., editor, *Neuroscience Databases. A Practical Guide*, chapter 9, pages 123–138. Kluwer Academic Publishers (Boston). 108, 131

Neal, R. M. (1993). Probabilistic inference using markov chain monte carlo methods. Technical report, University of Toronto Department of Computer Science. 9

Nessler, B., Pfeiffer, M., and Maass, W. (2010). STDP enables spiking neurons to detect hidden causes of their inputs. In *Proc. of NIPS 2009: Advances in Neural Information Processing Systems*, volume 22, pages 1357–1365. MIT Press. 23, 34

Nikolić, D., Haeusler, S., Singer, W., and Maass, W. (2007). Temporal dynamics of information content carried by neurons in the primary visual cortex. In *Proc. of NIPS 2006, Advances in Neural Information Processing Systems*, volume 19, pages 1041–1048. MIT Press. 77

Obermayer, K. and Blasdel, G. G. (1993). Geometry of orientation and ocular dominance columns in monkey striate cortex. *J Neurosci*, 13(10):4114–29. 115

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20. 109, 121

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan-Kaufmann, San Francisco, CA. 8

Pecevski, D., Natschläger, T., and Schuch, K. (2009). PCSIM: a parallel simulation environment for neural circuits fully integrated with Python. *Frontiers in Neuroinformatics*, 3(0). 52

Pérez, F. and Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29. 128

Pfister, J.-P., Toyoizumi, T., Barber, D., and Gerstner, W. (2006). Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural Computation*, 18(6):1318–1348. 56, 103

Plesser, H., Eppler, J., Morrison, A., Diesmann, M., and Gewaltig, M.-O. (2007). Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. *Lecture Notes in Computer Science*, 4641:672–681. 108

Raichle, M. E. (2010). Two views of brain function. *Trends in Cognitive Sciences*, 14(4):180–190. 36

Rao, R. P. and Ballard, D. H. (1999). Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nat Neurosci*, 2(1):79–87. 34

Rao, R. P. N. (2004). Bayesian computation in recurrent neural circuits. *Neural Computation*, 16(1):1–38. 34

Rao, R. P. N. (2007). Neural models of Bayesian belief propagation. In Doya, K., Ishii, S., Pouget, A., and Rao, R. P. N., editors, *Bayesian Brain.*, pages 239–267. MIT-Press, Cambridge, MA. 34, 35

Rao, R. P. N., Olshausen, B. A., and Lewicki, M. S. (2002). *Probabilistic Models of the Brain*. MIT Press. 8

Ray, S. and Bhalla, U. S. (2008). PyMOOSE: Interoperable scripting in Python for MOOSE. *Front. Neuroinform.*, 2(6). 132, 134

Reynolds, J. N., Hyland, B. I., and Wickens, J. R. (2001). A cellular mechanism of reward-related learning. *Nature*, 413:67–70. 56

Reynolds, J. N. and Wickens, J. R. (2002). Dopamine-dependent plasticity of corticostriatal synapses. *Neural Networks*, 15(4-6):507–521. 56

Schrauwen, B. and Campenhout, J. V. (2003). BSA, a fast and accurate spike train encoding scheme. In *Proceedings of the International Joint Conference on Neural Networks*, volume 4, pages 2825–2830. 98

Schultz, W. (2007). Behavioral dopamine signals. *Trends in Neuroscience*, 30:203–210. 56

Sejnowski, T. J. (1987). Higher-order Boltzmann machines. In *AIP Conference Proceedings 151 on Neural Networks for Computing*, pages 398–403, Woodbury, NY, USA. American Institute of Physics Inc. 35

Shi, L. and Griffiths, T. (2009). Neural implementation of hierarchical Bayesian inference by importance sampling. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 1669–1677. MIT Press, Cambridge, MA. 34

Shulz, D. E., Ego-Stengel, V., and Ahissar, E. (2003). Acetylcholine-dependent potentiation of temporal frequency representation in the barrel cortex does not depend on response magnitude during conditioning. *J Physiol Paris*, 97(4–6):431–439. 56

Shulz, D. E., Sosnik, R., Ego, V., Haidarliu, S., and Ahissar, E. (2000). A neuronal analogue of state-dependent learning. *Nature*, 403(6769):549–553. 56

Siegelmann, H. T. and Holzman, L. E. (2010). Neuronal integration of dynamic sources: Bayesian learning and Bayesian inference. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(3):037112. 34, 35

Silberberg, G., Bethge, M., Markram, H., Pawelzik, K., and Tsodyks, M. (2004). Dynamics of population rate codes in ensembles of neocortical neurons. *J Neurophysiology*, 91(2):704–709. 76

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive hebbian learning through spike-timing dependent synaptic plasticity. *Nature Neuroscience*, 3:919–926. 63

Steimer, A., Maass, W., and Douglas, R. (2009). Belief propagation in networks of spiking neurons. *Neural Computation*, 21(9):2502–2523. 34

Stevens, C. F. and Zador, A. M. (1998). Input synchrony and the irregular firing of cortical neurons. *Nature Neuroscience*, 1:210–217. 76

Thiel, C. M., Friston, K. J., and Dolan, R. J. (2002). Cholinergic modulation of experience-dependent plasticity in human auditory cortex. *Neuron*, 35(3):567–574. 56

Tolhurst, D., Movshon, J., and Dean, A. (1983). The statistical reliability of signals in single neurons in cat and monkey visual cortex. *Vision Research*, 23(8):775 – 785. 36

Toussaint, M. and Goerick, C. (2010). A Bayesian view on motor control and planning. In Sigaud, O. and Peters, J., editors, *From motor to interaction learning in robots. Studies in Computational Intelligence*, pages 227–252. Springer. 8

Verstraeten, D., Schrauwen, B., Stroobandt, D., and Campenhout, J. V. (2005). Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528. 98, 101

von Melchner, L., Pallas, S. L., and Sur, M. (2000). Visual behaviour mediated by retinal projection directed to the auditory pathway. *Nature*, 404:871–876. 2

Vul, E. and Pashler, H. (2008). Measuring the crowd within: Probabilistic representations within individuals. *Psychological Science*, 19(7):645–647. 36

Williams, S. R. and Stuart, G. J. (2002). Dependence of EPSP efficacy on synapse location in neocortical pyramidal neurons. *Science*, 295(5561):1907–1910. 28, 33

Williams, S. R. and Stuart, G. J. (2003). Voltage- and site-dependent control of the somatic impact of dendritic ipsps. *J Neurosci*, 23(23):7358–7367. 33

Yassin, L., Benedetti, B. L., Jouhanneau, J.-S., Wen, J. A., Poulet, J. F. A., and Barth, A. L. (2010). An embedded subnetwork of highly active neurons in the neocortex. *Neuron*, 68:1043–1050. 37

Yu, A. J. and Dayan, P. (2005). Inference, attention, and decision in a Bayesian neural architecture. In Saul, L., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 1577–1584. MIT Press, Cambridge, MA. 34