Graz University of Technology

Master's Thesis

The role of inhibitory networks in the self-organization of cortical microcircuits

Author:
Stefan Grabuschnig

Supervisor:
Assoc. Prof. DI Dr. Robert Legenstein

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

Arbeitsgruppe für Neuronale Informationsverarbeitung und Maschinelles Lernen
des Instituts für Grundlagen der Informationsverarbeitung
Institute for Theoretical Computer Science

October 2014

EIDESSTATTLICHE ERKLÄRUNG

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

Graz, am …………………………… ………………………………………………..
(Unterschrift)

STATUTORY DECLARATION

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

……………………………. ………………………………………………..
date (signature)

GRAZ UNIVERSITY OF TECHNOLOGY

Abstract

Faculty of Computer Science
Institute for Theoretical Computer Science

Master of Science

The role of inhibitory networks in the self-organization of cortical microcircuits

by Stefan Grabuschnig

A putative winner-take-all microcircuit motif found in layer 5 of the neocortex is subject to many modeling approaches. Mutual inhibition of competing excitatory neurons plays a crucial role in the functioning of these networks. This work investigates the effects of biologically plausible inhibition via interneurons in a network of spiking neurons. Previous work from Stefan Häusler models a network containing two different types of inhibitory interneurons, Basket Cells and Martinotti Cells. This model was adapted by incorporation of a threshold based bursting mechanism into its excitatory pyramidal cells, with the property of a delayed temporary threshold decrease triggered by action potentials of the respective neuron. Inhibition exerted by Martinotti Cells prevents this threshold decrease. This relation enables efficient control over learning processes in the network ultimately resulting in significantly improved performance in unsupervised pattern learning and self-organization.

TECHNISCHE UNIVERSITÄT GRAZ

Zusammenfassung

Fakultät für Informatik
Institut für Grundlagen der Informationsverarbeitung

Master of Science

The role of inhibitory networks in the self-organization of cortical microcircuits

von Stefan GRABUSCHNIG

Ein mutmaßliches winner-take-all Netzwerkmotiv in Layer 5 des Neocortex ist Gegenstand vieler Modellierungsansätze. Gegenseitige Inhibierung kompetierender erregender Neuronen spielt eine Schlüsselrolle in der Funktionsweise solcher Netzwerke. Im Rahmen dieser Arbeit wird die Auswirkung biologisch plausibler Inhibierung durch Interneuronen in einem Netzwerk mit spikenden Neuronen untersucht. Ausgangspunkt ist eine Arbeit von Stefan Häusler, welche ein Netzwerk mit zwei verschiedenen Typen inhibitorischer Interneuronen, Basket Zellen und Martinotti Zellen, modelliert. Dieses Model wurde durch Einbau eines schwellwertgesteuerten Burst-Mechanismus in die erregenden Pyramidenneuronen erweitert, welcher die Eigenschaft hat, dass der Schwellwert verzögert und vorübergehend durch ein Aktionspotential der jeweiligen Zelle herabgesetzt wird. Inhibierung durch Martinotti Zellen kann diese Herabsetzung verhindern. Dieser Zusammenhang ermöglicht effiziente Kontrolle über Lernprozesse innerhalb des Netzwerks, was in signifikant verbesserter Performance im Bezug auf Selbst-Organisation bei unüberwachtem Lernen und Erkennen von Mustern resultiert.

Contents

List of Figures

List of Tables

Abbreviations

AP	**A**ction **P**otential
bAP	**b**ackpropagating **A**ction **P**otential
BAC firing	**B**ackpropagation **A**ctivated **C**alcium spike firing
BC	**B**asket **C**ell
EM	**E**xpectation **M**aximization
EPSC	**E**xcitatory **P**ost-**S**ynaptic **C**urrent
EPSP	**E**xcitatory **P**ost-**S**ynaptic **P**otential
eTM Model	**e**xtended **T**sodyks **M**arkram Model
GABA	**G**amma-**A**mino-**B**utyric **A**cid
Hz	**H**ert**z**
IP	**I**ntrinsic **P**lasticity
IPSC	**I**nhibitory **P**ost-**S**ynaptic **C**urrent
IPSP	**I**nhibitory **P**ost-**S**ynaptic **P**otential
ISI	**I**nter-**S**pike **I**nterval
LBC	**L**arge **B**asket **C**ell
LIF Model	**L**eaky **I**ntegrate and **F**ire Model
LTD	**L**ong **T**erm **D**epression
LTP	**L**ong **T**erm **P**otentiation
MC	**M**artinotti **C**ell
ML	**M**aximum **L**ikelihood
mm	**m**illi**m**etre
μ**m**	micro**m**etre
MΩ	**M**ega-Ohm
mV	**m**illi-**V**olt
ms	**m**illi**s**econd

nA	**n**ano-**A**mpere
NBC	**N**ested **B**asket **C**ell
nm	**n**ano**m**etre
NMDA	**N**-**M**ethyl **D**-**A**spartate
PC	**P**yramidal **C**ell
SBC	**S**mall **B**asket **C**ell
SEM	**S**pike based **E**xpectation **M**aximization
STD	**S**hort **T**erm **D**epression
STDP	**S**pike-**T**iming dependent **P**lasticity
STP	**S**hort **T**erm **P**otentiation
VM	Java **V**irtual **M**achine
WTA	**W**inner-**T**ake-**A**ll

Introduction

Computer models of neural circuits have come a long way since the emergence of the first artificial neural networks. Biologically plausible spiking neuron models used in computational neurosciences are aimed at obtaining insights in neural information processing while additionally providing inspiration for new machine learning concepts. The manifold approaches in this field of research vary considerably in architecture and level of detail.

The models introduced in this work focus on a putative circuit motive found in layer V of the neocortex, where excitatory pyramidal neurons and different types of interneurons are thought to participate in a so called soft winner-take-all network. In this type of circuit excitatory neurons compete with each other via exertion of direct or indirect inhibition until a winner emerges for a certain network input.

In order to investigate the way interneurons orchestrate self-organization and learning in such circuits a model of Stefan Häusler [31] was used as a starting point. This model already includes biologically plausible inhibition using two distinct types of inhibitory interneurons. Basket Cells and Martinotti Cells are excited by the spike output of excitatory pyramidal neurons via synaptic connections featuring either depressing or facilitating short term plasticity [25], [22]. Inspired by recent work of Mathew Larkum et. al [30], [32], [33], [34] a bursting mechanism was incorporated into the excitatory neurons of this model. This was motivated by the idea of providing distinguishable spike coding addressing either facilitating synaptic connections in the case of high frequent bursts of spikes or depressing synaptic connections in the case of normal non-bursting spiking activity. An important detail is comprised by spike-timing dependent plasticity being only triggered by bursts of spikes [28], which is in contrast to the original model, where every spike of an excitatory neuron leads to adjustment of the weights of its synaptic connections to input units.

The burst mechanism was designed to be operated by a dynamic threshold, where a preceding action potential of the respective excitatory neuron temporarily decreases the threshold after a delay of 3 to 7 milliseconds. Inhibitory input of Martinotti Cells is able to prevent this effect.

In a task testing the model's ability for self-organization via unsupervised learning of two sets of patterns with different probabilities to be presented to the network, this leads surprisingly to a significantly increased performance in comparison to the original model. The threshold decrease mechanism provides the basis for the inhibitory populations to act in a quasi ideal fashion. Excitatory neurons specialized for a distinct pattern preempt unspecialized competitors in bursting. Their instantly fired burst of spikes excites Martinotti Cells, which in turn hinder other neurons from picking up a pattern by prevention of the threshold decrease.
A second feature learning task implements the popular Bars Problem introduced by Peter Földiak in 1990 [40]. This work's results show that the applied learning rule for spike-timing dependent plasticity determines how models internally derive and represent knowledge from their input. In this task the SEM learning rule [14], [16] fails to derive common features from different patterns, while a soft winner-take-all learning rule from [42] performs significantly better.
Altogether the results of this work suggest that details at the level of individual cell types have an enormous influence on the behavior and computational probabilities of neural microcircuits, supposedly being crucial for cortical information processing.

Overview

This thesis is organized in five chapters, where in a first chapter the theoretical background of this work is explained. Starting with basic concepts of spiking neuron models, such as the leaky integrate-and-fire model, the topic evolves stepwise accompanied by underlying biological and mathematical backgrounds. The motivation behind a winner-take-all network motifs is introduced, providing the base for a subsequent introduction of models performing spike-based expectation maximization and discussion of different realizations of inhibition in these models. At the end of the chapter the biological background for the modeling of biologically plausible inhibition is introduced.
The second chapter gives a detailed explanation of the model of Stefan Häusler, where the basic units representing different cell types are described as well as their communication via synaptic connections. The second part of the chapter is devoted to the adaptations of the model performed in the context of this work.
Experimental setup, tasks and results are explained in the third chapter, while the fourth chapter contains information about the evaluation and illustration of results.
Discussion of results as well as aspects concerning future and related work can be found in the last chapter. Two appendices contain detailed information about the implementation and usage of the simulation framework.

Chapter 1

Theoretical Background

1.1 Spiking Neuron Models

The first section deals with the mathematical concept of a spiking neuron model in the context of computational neurosciences, where modeling approaches at the level of neurons and small populations of neurons are used in order to understand principles of information processing in the human brain. Theoretical notions of basic elements (neurons, synapses, spikes) and formal spiking models are introduced and explained to the extent as they are necessary to understand the models introduced in this work. For this purpose some assorted chapters from Wulfram Gerstner's and Werner Kistler's book "Spiking Neuron Models" [1] are summarized in the following subsections.

1.1.1 Elementary Notions

The central information processing units in the brain are neurons forming a dense network. These are specialized cells communicating with each other via electrical and chemical signals. Their input is received via dendrites being long branching extensions of the cell body (soma). The elementary units of signal transmission are spikes (or action potentials (AP)), electrical pulses with an amplitude of 200 mV and a duration of 1 - 2 ms, being generated at the axonal hillock. The hillock is an exposed site of the cell body with a high density of sodium channels and leads into the axon, another long extension of the cell body connecting the neuron to the dendrites of other neurons.
Figure 1.1 illustrates a reconstruction of a layer 5 pyramidal neuron from mouse neocortex [2]. The prominent features of this type of neuron are a pyramidal shaped cell body and the large apical dendrite having a long straight trunk terminating in a complexly branched tuft. The smaller dendritic compartments extending from the soma are called

basal dendrites.

Signals are transmitted via synapses at the sites where the axon of the presynaptic neuron makes contact with the dendrite of the postsynaptic neuron. Synapses refer to bipartite bud-like structures located at spiny extensions of the dendrite and axon. The presynaptic part of the synapse releases neurotransmitters via exocytosis of vesicles into a narrow synaptic cleft of about 20 nm. The binding of these neurotransmitters to receptors at the postsynaptic surface in the synaptic cleft triggers a postsynaptic current. Excitation via synaptic currents received by a neuron leads to an increase in electric potential at the cytoplasmatic membrane of the neuron. If the neuron is excited sufficiently, this membrane potential will evoke an action potential at the axonal hillock. The level of detail at which this process is represented in spiking neuron models goes from simple point neuron models summarizing over synaptic input to detailed multi compartmental models where the membrane potential is calculated for each compartment at any discrete point in time.

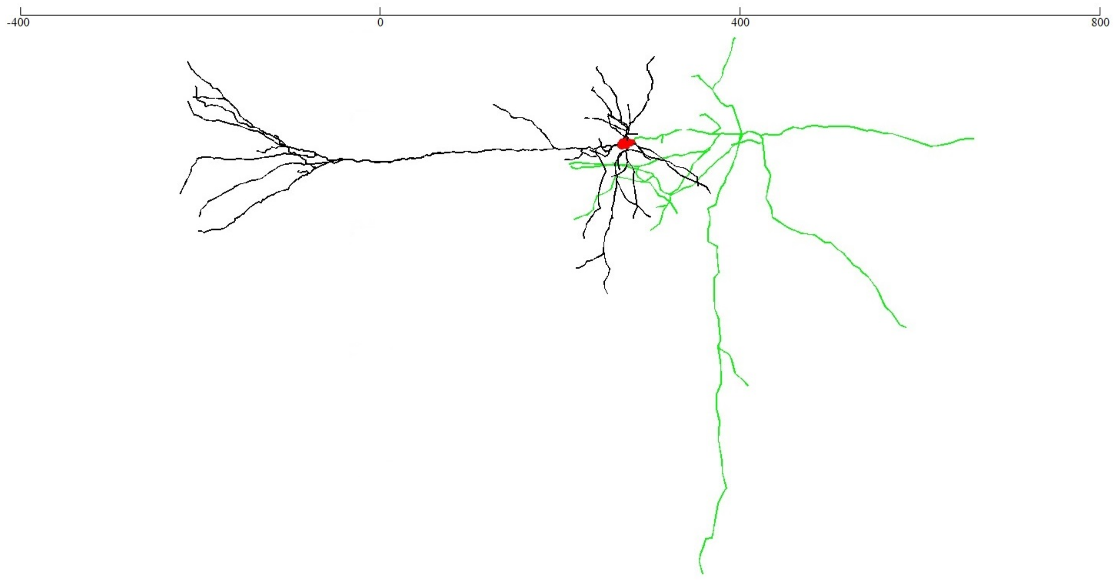

Figure 1.1: Layer 5 pyramidal neuron from mouse neocortex [2]. The dendritic compartments are colored black, the soma is colored red while the axon is green. The axis-unit is μm.

1.1.2 Leaky Integrate-and-Fire Model

In the leaky integrate-and-fire (LIF) model the phospholipid bilayer membrane of a neuron is interpreted as a parallel circuit of a membrane resistivity R_m and a capacitance C_m being comprised by the membrane's surface. The equivalent circuit of the model is shown in Figure 1.2. A driving current $I_\mathrm{m}(t)$ charges the potential $u(t)$ at the membrane and is split into a leak current $I_\mathrm{R}(t)$ and a current $I_\mathrm{C}(t)$ charging the capacitance. The current $I_\mathrm{m}(t) = I_\mathrm{R}(t) + I_\mathrm{C}(t)$ is given by

$$I_\mathrm{m}(t) = \frac{u(t)}{R_\mathrm{m}} + C_\mathrm{m} \cdot \frac{du}{dt}, \tag{1.1}$$

where introduction of a membrane time-constant $\tau_\mathrm{m} = R_\mathrm{m} \cdot C_\mathrm{m}$ leads to the standard form of the leaky integrator

$$\tau_\mathrm{m} \cdot \frac{du}{dt} = -u(t) + R_\mathrm{m} \cdot I_\mathrm{m}(t). \tag{1.2}$$

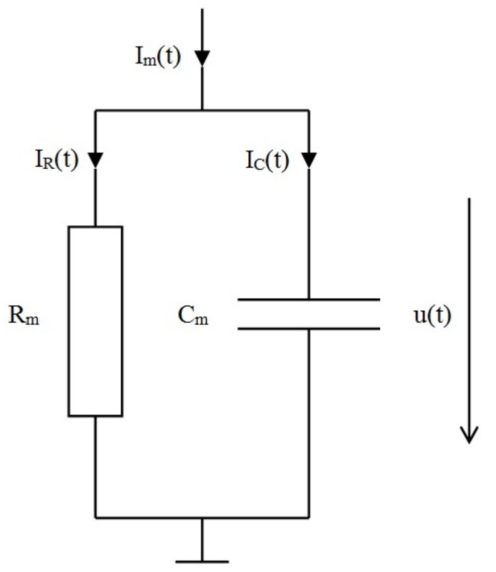

FIGURE 1.2: The formal circuit of the LIF model consists of a membrane resistivity R_m in parallel to a capacitance C_m. The membrane potential $u(t)$ is charged by a driving current $I_\mathrm{m}(t)$ and decays exponentially via a leak current I_R if no driving current is present.

If stimulated by a constant current I_0 starting at a time $t^{(0)}$, the membrane potential's time course can be formulated as

$$u(t) = R_\mathrm{m} \cdot I_0 \cdot \left(1 - e^{-\frac{t - t^{(0)}}{\tau_\mathrm{m}}}\right), \tag{1.3}$$

being the solution of Equation 1.2 with the initial condition that $u(t^{(0)})$ is zero.
Note that the original LIF model as described in [1] is gated by a threshold ϑ, where a spike is fired the moment $u(t)$ reaches this threshold. The neurons used by the model described in Section 2.1 are stochastic point neurons whose spiking probabilities exponentially depend on $u(t)$.

1.1.2.1 Synaptic Input

If the LIF neuron is not viewed apart from but as part of a network of neurons, the driving current is induced by input spikes of presynaptic neurons j generating current pulses. In this context the input current $I_i(t)$ of unit i is formulated as a linear sum of postsynaptic current pulses given by

$$I_i(t) = \sum_j w_{ij} \sum_f \alpha(t - t_j^{(f)}), \tag{1.4}$$

where w_{ij} denotes the efficacy of the synaptic connection from unit j to unit i and f denotes a presynaptic spike fired at a time $t_j^{(f)}$. The kernel function $\alpha(t - t_j^{(f)})$ describes the shape and time course of a current pulse. The parameter w_{ij}, also called weight, determines the amplitude of the postsynaptic response to a presynaptic spike.

1.1.3 Spike Trains

The output of a single neuron can be represented by binary values in a discretized time space, where 1 symbolizes a spike at a discrete point in time and 0 the absence of a spike. The modeling of spike trains depends on the underlying theory of neural coding varying from precise temporal location of each spike to stochastic processes [3]. If spike trains are subjected to statistical treatment regarding inter-spike intervals (ISI), their generation is related to the mathematical concept of a stochastic point process. If in this process "events" (spikes) are indistinguishable from each other except for their time of occurrence, the ISI can be viewed as a random variable being drawn from an underlying probability distribution. If this distribution does not change over time, the process is stationary [4]. This leads to the assumption that spike trains can be sufficiently well described by a Poisson process, where the time a spike is fired does not depend on a previous spike. This approach neglects the fact that this assumption does not hold regarding biological observations such as refractoriness and bursts of spikes [5].
In the Poisson model of spike generation [6] the level of excitation of a neuron directly results in an average output firing rate $r(t)$ at which spikes are generated. If this driving signal is constant over time this is referred to as homogeneous Poisson process.
A commonly used procedure for numerical generation of Poisson spike trains is based on the approximation of the spike probability $p_{\text{spike}}([t, t + dt])$ within the time interval from t to $t + dt$

$$p_{\text{spike}}([t, t + dt]) \approx r(t) \cdot dt, \tag{1.5}$$

for sufficiently small dt.

1.1.4 Synaptic Plasticity

Learning in neuronal circuits is thought to occur for the most part via synaptic plasticity referring to a structural change of the synapse leading to growth or shrinkage. With this comes a change in synaptic efficacy, which is characterized by the synaptic weight parameter w_{ij} as introduced in Equation 1.4. The signal triggering this process is the back propagation of action potentials via the dendrites. With this signal synapses are able to relate presynaptic excitation to postsynaptic spike firing. Depending on the relative timing of a presynaptic spike at $t_j^{(f)}$ and a postsynaptic AP at $t_i^{(f)}$ the synaptic connection is strengthened or weakened, being referred to as long term potentiation (LTP) or long term depression (LTD).
In mathematical models learning refers to a process relating the expectation value of the weight vector to the statistical properties of presynaptic and postsynaptic spike trains. The procedure of adjusting the weights during the learning process is being referred to as learning rule. Spike-timing depended plasticity (STDP) models plasticity at the level of individual spikes relating pre- and postsynaptic firing. Using this learning rule a model acts as coincidence detector. If a set of presynaptic spikes triggers an action potential, the synapses receiving those spikes with a negative spike timing $t_j^{(f)} - t_i^{(f)}$are strengthened, while positive spike timing leads to depression of the respective synaptic connection. This leads to the introduction of a learning window $W(s)$ determining the change of synaptic weights $\Delta w_{ij} \propto W(t_j^{(f)} - t_i^{(f)})$. An example for a learning window is given by

$$W(s) = \begin{cases} A^{+} \cdot \exp\left[\frac{s}{\tau_1}\right] & \text{for } s < 0 \\ A^{\text{-}} \cdot \exp\left[\frac{-s}{\tau_2}\right] & \text{for } s > 0 \end{cases}, \tag{1.6}$$

where the parameters A^{+} and A^{+} define the amplitude of the weight change and the time constants τ_1 and τ_2 define the shape of the window illustrated in Figure 1.3.

1.2 Cortical Microcircuits

The computational power and manifold sophisticated abilities of human and animal brains have motivated intensive research for decades. Up to now many advances were made in this concern, but the enormous complexity of this subject still proves to be inscrutable and enigmatic in many aspects. While understanding neuronal computation at a high level involving several brain areas seems not to be within reach in the near

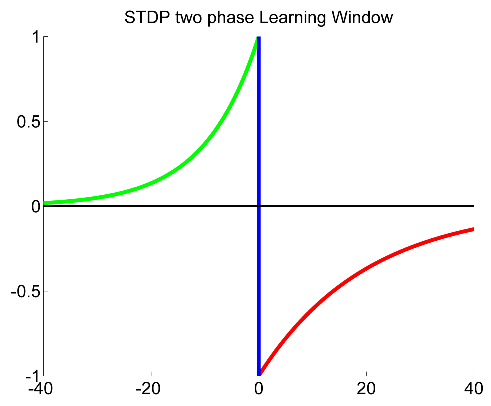

FIGURE 1.3: STDP learning window for Equation 1.6 with $A^{+} = 1$, $A^{-} = -A^{+}$, $\tau_1 = 10$ ms and $\tau_2 = 20$ ms.

future, the question arises how one can break down this problem to the level of reoccurring canonical circuit motifs. The extent at which neocortical circuits can be considered canonical is still unclear [7], nevertheless there exist many different models of circuit motifs being based on experimental data [8].

1.2.1 Organization and Structure of the Neocortex

The neocortex is the largest part of the human cerebral cortex covering its hemispheres. Being the most developed of cortical tissues it is involved in higher functions such as conscious thought, spatial reasoning, motor control, sensory perception and language. With a thickness of 3 to 4 mm it contains around 10^{10} neurons and about the same amount of glial cells, where roughly 80 % of the neurons are comprised by excitatory pyramidal cells and the remaining 20 % are inhibitory interneurons. Neurons connect to each other within the neocortex and to other brain areas via a vast number of approximately 10^{12} synapses. It is widely agreed on a horizontal organization of the neocortex into six laminae, labeled by roman numerals from I to VI from outside to inside. A vertical columnar organization is still discussed, where the basic unit is thought to be the minicolumn, a narrow chain of 80 to 100 neurons (except for striate cortex where the amount is about 2.5 times higher) extending vertically through all cortical layers. Cortical columns are considered to be local modular processing units consisting of many minicolumns bound together [9].
The Review "Neural Circuits of the Neocortex" of Rodney J. Douglas and Kevan A.C. Martin [8] introduces models for laminae specific projections and connectivity patterns being universal for all neocortical areas. Figure 1.4 shows how the chain of information processing within the neocortex could look like. The model for neocortical computation

evolves around layer V pyramidal cells, where information processing is thought to converge. Input from the thalamus enters the circuit in layer IV, where excitatory neurons project to superficial layers, mostly to layer III. Excitatory neurons in superficial layers II and III arborize extensively within their layer, forming intralaminar connections. Also inhibitory interneurons are more densely distributed over those layers. Interlaminar connections from layer III terminate mainly in layer V but also feedforward projections to other cortical areas originate in layer III. Layer V projects to layer VI which in turn projects to the thalamus. Feedback connections originate in layers V and VI terminating in superficial layers outside of layer IV. A special feature of the neocortex is layer I being called its "crowning mystery". Consisting mostly out of the distal tufts of pyramidal cells and a few interneurons layer I is the major target for feedback input.

Excitation of pyramidal cells can be adjusted by inhibitory input from interneurons. Morphology of interneurons is conserved, where ten types of interneurons are distinguished being responsible for different types of inhibitory functions. About half of the interneurons are Basket Cells, targeting specially somata and proximal dendrites of pyramidal neurons as well as other interneurons [10]. Different types of Basket Cells are thought to deliver lateral inhibition to their direct vicinity as well as to neighboring and distant cortical columns. It is believed that pyramidal cells and inhibitory neurons participate in a network motif that realizes a soft winner-take-all circuit, where a winner suppresses the output of its local competitors. Different variants of those circuits are often used in many neural network models. The models introduced in Chapter 2 are implementations of such a winner-take-all network motif within layer V operating on feedforward input from layer III.

1.2.2 Winner-Take-All Network Motifs

The simplest implementation of a competitive-learning winner-take-all circuit with n output units (WTA$_n$) is the hard winner-take-all network motif. Binary gates map their input, computed as weighted linear sum, to an output with the constraint that exactly one of the n output units can be 1 at a distinct point in time. Therefore the strongest unit inhibits all other units with lower levels of activation and forces them to be zero. In soft WTA modules, the output is comprised by n analog variables, defining the rank of the respective units.

In more biologically plausible approaches the input and output of the model is encoded by firing rates. All units inhibit each other laterally depending on their output rate, where the most active unit exerts the strongest inhibition on its competitors, ultimately stopping them from firing. In the end only one winner-unit is firing for a certain input being called "expert" for this input or class of inputs. After learning, such models

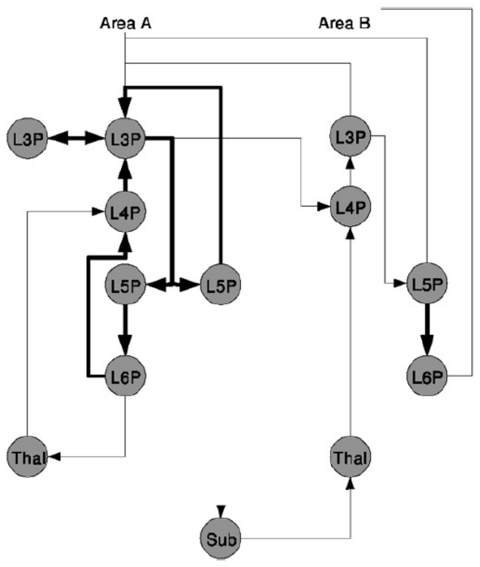

FIGURE 1.4: Schematic illustration of dominant interactions between excitatory pyramidal cells in their respective layers from the review "Neural Circuits of the Neocortex" [8]. Thick arrows indicate directed interactions within a local processing module called "patch" [8]. Interactions with thalamic and other cortical areas are indicated by thin arrows.

produce one or a set of experts, each for a set of input-patterns or classes of input patterns depending on the level of generalization over the respective learning task.
In "On the Computational Power of Winner-Take-All" [11] Wolfgang Maass gives an detailed comparison of the properties and computational power between winner-take-all motifs and well established concepts in machine learning such as the perceptron [12]. Biologically plausible implementations of a winner-take-all motif come with an important restriction. While in classical artificial neuron models the weights of the connections to input units can be either positive or negative, a biological plausible implementation can only be subject to excitatory feedforward input. Therefore only positive weights are possible, while the lateral inhibition can only be negative. Wolfgang Maas shows that this restriction does not come with a loss of computational power. The same holds for a restriction where synaptic plasticity, or more generally the adaption of weights, only occurs for excitatory feedforward connections but not for lateral inhibitory connections. It turns out that the computational power of a hard winner-take-all circuit is that of a multi-layer perceptron. More generally the full computational power of two layers of threshold gates [13] can be a achieved by a single WTA$_n$ network, while vice versa it takes n^2 threshold gates to realize the function of WTA$_n$. In case of a soft winner-take-all network, the computational power is that of an universal approximator for any continuous function.

1.3 Spike-based Expectation Maximization

A class of biological plausible realizations of a winner-take-all circuit is the result of recent work [14], [15], [16], [17] summed up under the term spike-based expectation maximization or in short SEM model. This type of model is shown to implicitly perform Bayesian computation through STDP. Comprising an implicit generative model [13] it is able to extract hidden causes from high dimensional data by performing an online stochastic version of expectation maximization (EM). Applied to pattern recognition tasks or inference of reliable hidden causes EM is considered to be one of the most powerful or even the most powerful theoretical framework for unsupervised learning. With this the SEM model is able to so solve demanding computational challenges such as the recognition of handwritten digits [17], [18].
It is a popular hypothesis that the brain creates and maintains an internal model of its environment by performing Bayesian inference on perceived impressions. Remarkable for the SEM model is the linkage of STDP, winner-take-all and EM, which could provide insight into organization and computation in cortical networks.

1.3.1 Bayesian Inference in Cortical Circuit Models

Inference usually refers to a process where information from a set of observations for a random variable a is used to infer a model for the probability density $p(a)$. If a conditionally depends on one or more other random variables, for example a random variable b, the process of linking information to model this dependencies in form of posterior probabilities $p(a|b)$ and joint probabilities $p(a,b)$ by applying Bayes' theorem

$$p(a|b) = \frac{p(a,b)}{p(b)} = \frac{p(b|a) \cdot p(a)}{p(b)} \tag{1.7}$$

is called Bayesian inference.
At the level of neural coding the value of a single variable is represented by the spiking activity of a population of neurons. The response of such a population is noisy and shows a significant trial-to-trial variability. How information being represented by a population, for example in sensory areas, can be decoded by downstream neurons is an open question but thought to occur via Bayesian computations. This problem can be formulated as relating the response of a population of N neurons $(y_1, \ldots, y_N)$ to a hidden cause Θ being not directly observable. The relation can be represented by the conditional probability distribution $p(y|\Theta)$. Inferring Θ from y takes place by applying

$$p(\Theta|y) \propto p(y|\Theta) \cdot p(\Theta). \tag{1.8}$$

In the SEM model this inference is performed autonomously via STDP comprising a type of mixture model [13]. Unsupervised learning of a sparse representation of most likely hidden causes for observations y happens through implicit inference of the distribution of y for the respective hidden cause. This distribution is represented by the synaptic weights learned by the model.
Mixture models define a framework for building complex probability distributions for latent variables and can also be used to cluster data. This type of generative model divides the data into subpopulations by assigning data points to specific components of the mixture, defining interpretations of latent variables. This can be done by finding likelihood estimators for latent variables by application of the expectation maximization algorithm.
In context of a winner-take-all motif this can be viewed in the following way. The model tries to find regularities in the spiking patterns being delivered to the model by input populations. This is done by assigning putative patterns to the output units of the WTA motif. By this a unit becomes an expert for a putative pattern by learning its probability distribution, simultaneously maximizing the likelihood of the pattern being attributable to a latent variable or hidden cause. The assignment step happens when the model choses a winner, being most probably the unit with the highest grade of excitation for a certain input. When the respective winner fires a spike, it draws a sample from the distribution of the assigned input. By adjusting its synaptic weights, it tunes the probability distribution represented by those weights towards the most likely distribution of its input.
The selection of winners within a model depends on its realization of lateral inhibition, having consequences on the internal representation of knowledge. In a hard winner-take-all model only one unit may be active at a time and therefore only one unit will represent a putative hidden cause. In soft winner-take-all implementations there can be more than one winner, where all winners contribute to an inhibitory signal reducing the probability for other units to pick up the same pattern. Such a model will converge to a state where there is sufficient inhibition exerted by winner units for every putative hidden variable. Different winners may reflect different phenotypes of a hidden cause, for example different variants of a handwritten digit.

1.3.2 The basic SEM-Model

The illustration in Figure 1.5 shows a schematic overview on the components of the SEM model. A hidden cause is responsible for a distinct configuration of external variables $(x_1, \ldots, x_M)$, resulting in generation of Poisson spike trains by the input units $(y_1, \ldots, y_n)$. Center of the model is a set of K competing z-units $(z_1, \ldots, z_K)$ being also the output units of the model. The z-units calculate their grade of excitation $u_k(t)$ via their respective synaptic weights $(w_{k1}, \dots, w_{kn})$ applied to the model's input in form of the linear sum

$$u_k(t) = w_{k0} + \sum_{i=1}^{n} w_{ki} y_n(t), \tag{1.9}$$

where w_{k0} is a bias and $y_n(t)$ depends on the respective realization of the spike input's synaptic integration.
Depending on how inhibition is realized in the respective implementation, the circuit can either be a hard or a soft winner take all circuit which reflects in the model's output. In [17] inhibition is modeled as selection process in a hard winner-take-all circuit, where every distinct point in time in a discretized timespace only one unit can be active. The active unit generates a Poisson spike train with a spike every 5 ms on average. The selection mechanism for the winner neuron at time-step t is modeled by the soft-max distribution

$$p(z_k|y) = \frac{e^{u_k(t)}}{\sum\limits_{l=1}^{K} e^{u_l(t)}}. \tag{1.10}$$

In the soft winner-take-all implementation of [14], there exists the possibility of two z-units spiking at the same time (becoming arbitrary small by choosing sufficiently small time-steps). The effect of lateral inhibition is modeled by a global inhibitory signal $I(t)$, where every output spike fired by a z-unit contributes to this signal. Therefore a unit with high excitation, firing many spikes, will suppress its competitors in the circuit. The stochastic spike firing is modeled by an exponential dependency of the spiking probability on the grade of excitation given by

$$p(z_k \; fires \; a \; spike \; at \; time \; t) \propto e^{u_k(t) - I(t)}. \tag{1.11}$$

A common feature of different implementations of the SEM model is that every time a z-unit fires a spike, it updates all its synaptic weights according to a generalized STDP learning rule being referred to as SEM learning rule.

1.3.3 The SEM STDP Learning Rule

The generalized formulation of the theoretical motivated SEM learning rule [15] is given by

$$\Delta w_{ki} = \eta \cdot z_k \cdot \left(\alpha \cdot e^{-w_{ki}} \cdot y_i(t) - 1 \right), \tag{1.12}$$

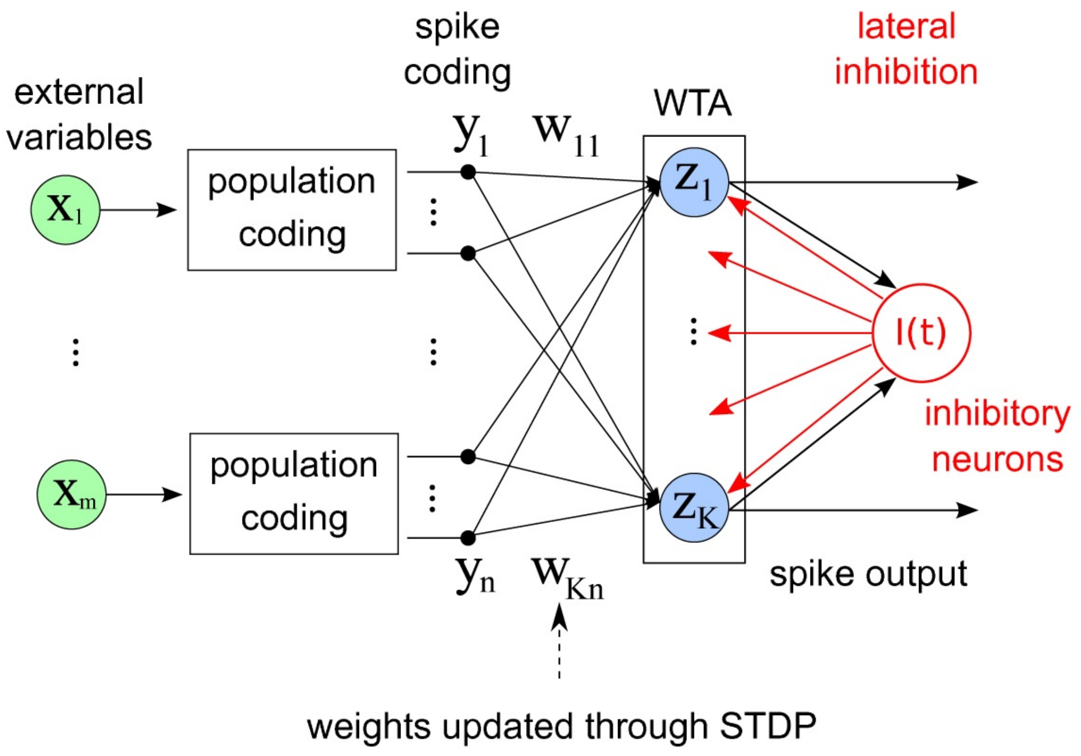

FIGURE 1.5: The SEM model as it is introduced by [14]. K competing z-units receive their feedforward input in form of Poisson spike trains via synaptic connections to the N input units, where each z-unit is connected to all input units. A global inhibitory signal affects all z-units equally.

where η denotes the learning rate, $z_k = 1$ indicates a spike fired by z_k at the respective update step and $z_k = 0$ the absence of a spike resulting in Δw_{ki} being zero. α, being a positive constant, is responsible for balancing potentiation and depression. The shape of the STDP window (see Section 1.1.4) resulting from this rule depends on the implementation dependent realization of $y(t)$. In any case $\alpha \cdot e^{-w_{ki}} - 1$ corresponds to a spike timing dependent LTP-window. The absence of a presynaptic spike before a postsynaptic spike leads to $y(t)$ being zero, resulting in LTP since the remaining non-zero term is $-\eta$.

This form of STDP has some important advantages while still fitting in the phenomenological framework of STDP rules. Stable equilibrium weight settings can be clearly interpreted by probability theory. The dependency of the level of potentiation on the weight before spike pairing is consistent with experimental studies as well as the independence of the amount of depression. Since synapses cannot grow infinitely in size, plasticity has to be going into saturation at a certain level of synaptic efficacy. Also the learning rule can be derived from the principles of adapting generative models to input statistics.

1.3.4 STDP performs Expectation Maximization

A generalized description of the EM algorithm can be found in Christopher M. Bishop's book "Pattern Recognition and Machine Learning" [13]. The algorithm comprises a powerful framework for finding maximum likelihood (ML) solutions for hidden variables, being applicable to a broad variety of different models. Goal of the algorithm is to derive knowledge about the distribution of hidden variables from a set of observations from random variables being (supposedly) conditionally dependent on the hidden variables. Applied to a mixture model, the algorithm starts with an arbitrary set of parameters defining the mixture. In a first step the most likely distribution of the hidden variables is estimated for the current setting of the model's parameters (E-Step). In the next step the algorithm tries to find a new configuration of the parameters in order to maximize the data log-likelihood in respect of the current estimation of the hidden variables (M-step). E-step and M-step are alternately repeated until the algorithm converges.
In [15] and [14] it is shown how STDP in a winner-take-all circuit with lateral inhibition can directly be related to the EM algorithm. Application of the SEM learning rule follows attractor dynamics in weight space, where attraction centers are weight settings being stable under the dynamics of the network. The equilibrium condition for a weight w_{ki} of the synaptic connection from input y_i to output unit z_k is given by

$$E\left[\Delta w_{ki}\right] = 0 \leftrightarrow w_{ki} = \log p(y_i|z_k\ fires) + \log c, \tag{1.13}$$

where $E\left[\Delta w_{ki}\right]$ denotes the value of the expected weight update and $p(y_i|z_k\ fires)$ is the probability for a presynaptic spike from input y_i given z_k firing a spike. This stochastic convergence results from the exponential dependency on the current weight in Equation 1.12 and corresponds to optimal weight settings from the perspective of a generative model. Using the learned weights of a z-unit to formulate the average rate of a Poisson process, the z-unit can generate spike trains corresponding to the input distribution of the respective hidden cause encoded by z_k. If all K hidden causes encoding z-units have the same probability to be active, the generative model is given by

$$p(\mathbf{y}|\mathbf{W}) = \frac{1}{K}\sum_{k=1}^{K}\prod_{i=1}^{N}\text{Poisson}(y_i, \alpha^{-1}e^{w_{ki}}), \tag{1.14}$$

where Poisson(y, λ) denotes the Poisson distribution over y with the "rate" λ.
Note that in contrast to the EM algorithm described at the beginning of this subsection learning in the SEM model does not make use of a whole set of observations $\mathbf{X}$ but rather of a single observation at a distinct point in time. When spiking for such an observation, the model performs the E-step of the algorithm in a stochastic way. The winner is drawn from a distribution based on knowledge of hidden causes modeled by

the weights of all z-units. The unit whose weights best resemble the observation has the highest chance to be selected. Applying the STDP learning rule of Equation 1.12, the weights then are tuned towards the observation, with this increasing the probability for the unit to spike for this same observation. This corresponds to the execution of the M-step of the EM algorithm. Like the EM algorithm, learning in the SEM model is comprised by two basic concepts. While STDP maximizes step by step the likelihood of a z-unit to represent an actual hidden cause, modeling the distribution of the related input spike trains, the lateral inhibition is crucial for estimating and selecting a winner. A problem addressed by the models introduced in Chapter 2 is the availability of the inhibitory signal in biologically plausible circuits. In contrast to lateral inhibition in the basic SEM model, this inhibitory signal does not simultaneously arrive with the input. Because inhibitory neurons have to be activated in a first step, the inhibition exerted by them on the z-units comes with a temporal delay after the input.

1.4 Modeling biologically plausible Inhibition

There are many open questions when relating computational models of cortical microcircuits to their biological counterparts in the brain, where this work's focus lies on the role of inhibition by two certain types of interneurons, Basket Cells and Martinotti Cells. The task of modeling biologically plausible inhibition via interneurons, like in the models introduced in Chapter 2, is due to the complex interplay of several types of neurons and synapses very challenging. The following subsection will provide a closer look onto the biological aspects that were taken into consideration for this approach.

1.4.1 Inhibitory Interneurons

The review ”Interneurons of the Neocortical Inhibitory System” from Henry Markram et. al [10] gives an overview on scientific insights on properties and function of the different types of interneurons being found in the neocortex. They vary greatly in morphology being particularly specialized in targeting of certain neuronal sub-domains, cortical layers or columns. A common feature of most mature interneurons are the aspiny dendrites distinguishing them from pyramidal cells. While not all interneurons are inhibitory, most of them are, using GABA (gamma-amino-butyric acid) as neurotransmitter. Classified by morphology there are different layer dependent compositions of interneurons as illustrated in Figure 1.6. Especially in layers V and VI different types of ”local circuit” Basket Cells and Martinotti Cells seem to be the dominant source of intralaminar inhibition. In addition to classification by morphology interneurons can be subdivided according to their ability to target different domains, distinguishing axon targeting, soma and proximal dendrites targeting, dendrite targeting and dendrite and tuft targeting interneurons.

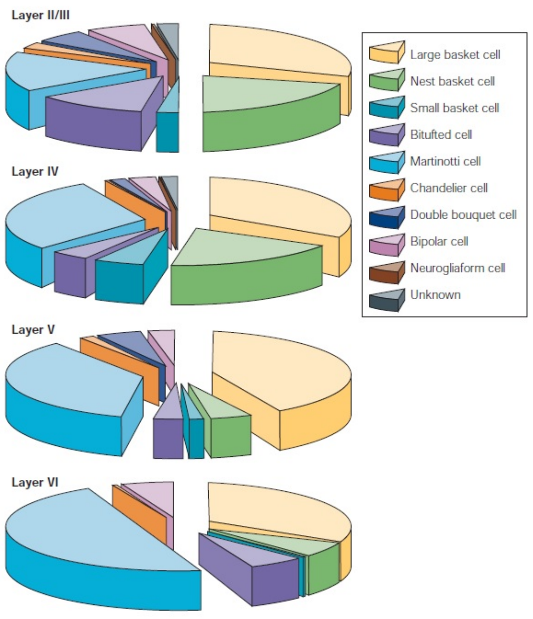

FIGURE 1.6: The graph from [10] shows the layer specific composition of different types of interneurons in the somatosensory cortex of juvenile rats.

1.4.1.1 Basket Cells

The major part of interneurons is constituted by Large Basket Cells (LBC) and Nested Basket Cells (NBC). The third of the three main subclasses is the Small Basket Cell (SBC). All of them target specifically the soma and proximal dendrites of pyramidal cells, with this being in the unique position to adjust the gain of the integral synaptic response. Basket cells can be distinguished from other types of interneurons in respect to immunoreactivity by their expression of two types of calcium binding proteins, parvalbumin and calbinidin.

LBCs are the "classic" type of Basket Cells, being the primary source of intralaminar lateral inhibition across columns. They typically feature sparse axonal arborization, while SBCs feature dense local arborization restricted to the same cortical column. The highest number of synapses on pyramidal cells is formed with SBCs. The size of NBCs, named after their birds' nest like appearance, is somewhere in between LBCs and SBCs.

1.4.1.2 Martinotti Cells

Martinotti Cells are found in all neocortical layers with the exception of layer I. Their axon projects mainly to layer I, where it inhibits the tuft dendrites of pyramidal cells, and also extends horizontally, inhibiting tuft dendrites of neighboring and distal columns. An unusual feature of Martinotti Cells is that they are not restricted to a single domain but rather target proximal dendrites and somata as well.
Interneurons targeting dendritic domains are in the position to influence dendritic processing and synaptic plasticity, either locally or by affecting backpropagation of action potentials. It is also shown that they have a significant effect on the generation of dendritic calcium spikes [19], [20].

1.4.2 Short Term Plasticity

In contrast to long term plasticity, as introduced in Section 1.1.4, lasting for hours or even weeks, another form of synaptic plasticity acts on a notable faster timescale lasting only for seconds. Being referred to as short term plasticity it is thought to comprise a feature in cortical information processing.
Two basic variants of short term plasticity are distinguished at the level of individual synapses. While depressing synapses attenuate transmission of high frequent information, the synaptic response of facilitating synapses is even amplified. Therefore the same presynaptic cell might transfer quite different information to postsynaptic neurons depending on the type of synapse connecting them. There exist typical cell type specific connectivity patterns for short term plasticity. While pyramidal cells are typically connected to Basket Cells via depressing synapses, Martinotti cells are excited by pyramidal cells via facilitating synapses [21]. The model for an inhibitory circuit in neocortex, being based on this properties, is illustrated in Figure 1.7.
A phenomenological model of short term plasticity was introduced by Misha Tsodyks and Henry Markram [22], [23], [24] being based on presynaptic effects such as vesicle depletion and accumulation of calcium in the presynaptic terminal, affecting utilization of synaptic efficacy by increasing the release probability of vesicles. The model is also being referred to as extended Tsodkys Markram Model (eTM-Model) [25] and is given by two differential equations for the number of vesicles $R(t)$ and release probability $o(t)$ which corresponds to synaptic utilization. Their dependency on a presynaptic action potential at a time t_{AP}, modeled by a Dirac delta distribution $\delta(t - t_{\mathrm{AP}})$, is given by

$$\frac{dR(t)}{dt} = \frac{1 - R(t)}{\tau_{\mathrm{dep}}} - o(t^-) \cdot R(t^-) \cdot \delta(t - t_{\mathrm{AP}}) \tag{1.15}$$

and
$$\frac{do(t)}{dt} = \frac{O - o(t)}{\tau_{\text{fac}}} + f\left[1 - o(t^-)\right] \cdot \delta(t - t_{\text{AP}}), \tag{1.16}$$

where τ_{dep} and τ_{fac} are depression and facilitation time-constants and $f\left[1 - u(t^-)\right]$ models the increase of release probability after a presynaptic spike, decaying back to the baseline probability O. t^- indicates that these functions should be evaluated in the limit approaching the time of the action potential from below [25].

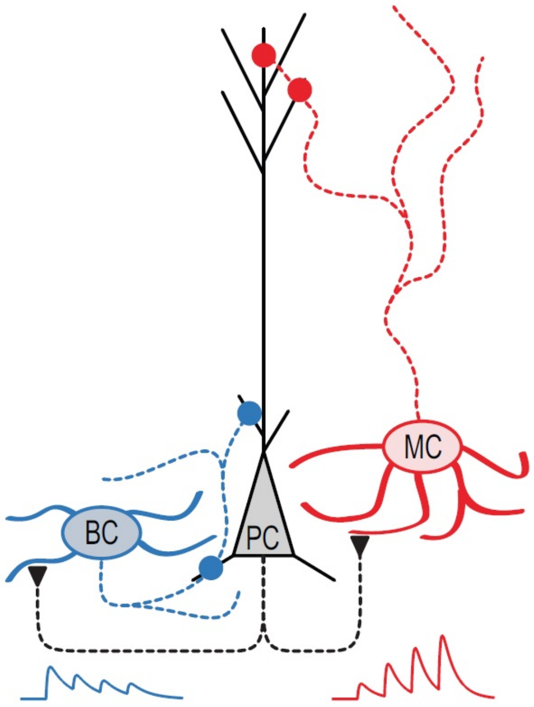

FIGURE 1.7: This schematic from [21] illustrates a model for inhibition in the neocortex. Frequency dependent excitation of Basket Cells and Martinotti Cells occurs via either depressing or facilitating synapses by presynaptic spike output of pyramidal cells. In turn Basket Cells exert inhibition on pyramidal cells targeting somata and proximal dendrites in their vicinity, while Martinotti Cells also target distal dendritic domains.

1.4.3 Layer 5 Pyramidal Cells

Pyramidal cells are primarily found in structures being responsible for advanced cognitive function. They have several characteristic features, such as relatively short basal dendrites compared to the large apical dendrite which bifurcates before connecting the soma to a complex tuft. Emanating from the trunk of the apical dendrite oblique dendrites extend at various angles. Although being characteristic, those features vary considerably between different layers and cortical regions [26].

Excitatory input from local sources arrives at the soma and proximal dendrites. The

tuft is the main target for feedback input, while inhibition is received for the most part at the soma and axon. Synaptic integration of input at the apical dendrite is a complex operation involving several nonlinearities. Polsky et. al [27] suggest a three layer model being shown in Figure 1.8. In a first layer N-methyl-D-aspartate (NMDA) receptors, a type of glutamate receptor at the postsynaptic terminal, act as a detector for synchronous firing by providing supralinear summation of coinciding synaptic responses if located in a vicinity within less than 100 μm [28] [29]. A second nonlinearity is comprised by a calcium spike initiation zone at the end of the apical trunk close to the tuft. The calcium spike produces long plateau-type potentials driving the pyramidal cell to fire a high frequent burst of spikes if triggered [30]. The third and last layer in this model is the somatic action potential initiation zone at the axonal hillock.
The role of complex dendritic mechanisms in controlling STDP is explained in the review of Björn Kampa et. al [28], suggesting that long term plasticity is triggered by bursts of spikes rather than single spikes.

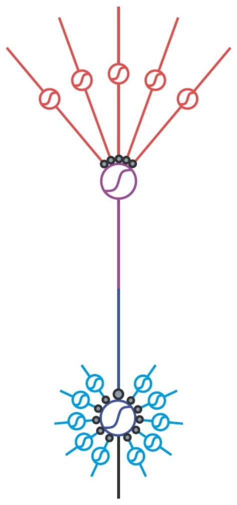

FIGURE 1.8: The three layer model for dendritic integration at pyramidal cells as introduced in [29]. Supralinear integration zones are depicted by sigmoidal gates. These nonlinearities are comprised by NDMA receptors in apical (red) and basal (blue) dendrites, the dendritic calcium spike initiation zone(magenta) and the somatic AP initiation zone at the axonal hillock.

Chapter 2

The Model

2.1 Original Model

This section describes the starting point of this work, a model of a cortical neural circuit winner-take-all (WTA) motif from Stefan Häusler [31]. It is based on previous work in the context of spike-based Expectation Maximization (SEM) from Stefan Habenschuss [16] and Bernhard Nessler [14].
The circuit is composed of a set of N input units $y_1, \ldots, y_N$, K output units $z_1, \ldots, z_K$, abstracting layer 5 pyramidal cells (PCs), and two populations of inhibitory interneurons abstracting Basket Cells (BCs) and Martinotti Cells (MCs). Figure 2.1 shows the connectivities of the network topology. The following subsections describe in short the individual units of the original model.

2.1.1 Input Layer

The input layer is constituted by N Poisson-Processes, delivering the feed-forward input in form of binary spike trains. Depending on every input unit's rate it is rolled out if there occurs a spike or not for every discrete time-step of size $dt = 1$ ms. For more details see Section 3.1.2.

2.1.2 z-Layer

The layer 5 pyramidal cells of the z-layer are represented by stochastically spiking point neuron models, where every unit's rate $r_k(t)$ depends on its membrane potential $u_k(t)$ being updated at every time-step. The membrane potential itself decays exponentially with a time-constant of $\tau_{\mathrm{m}} = 20$ ms and is recharged by an excitatory post-synaptic

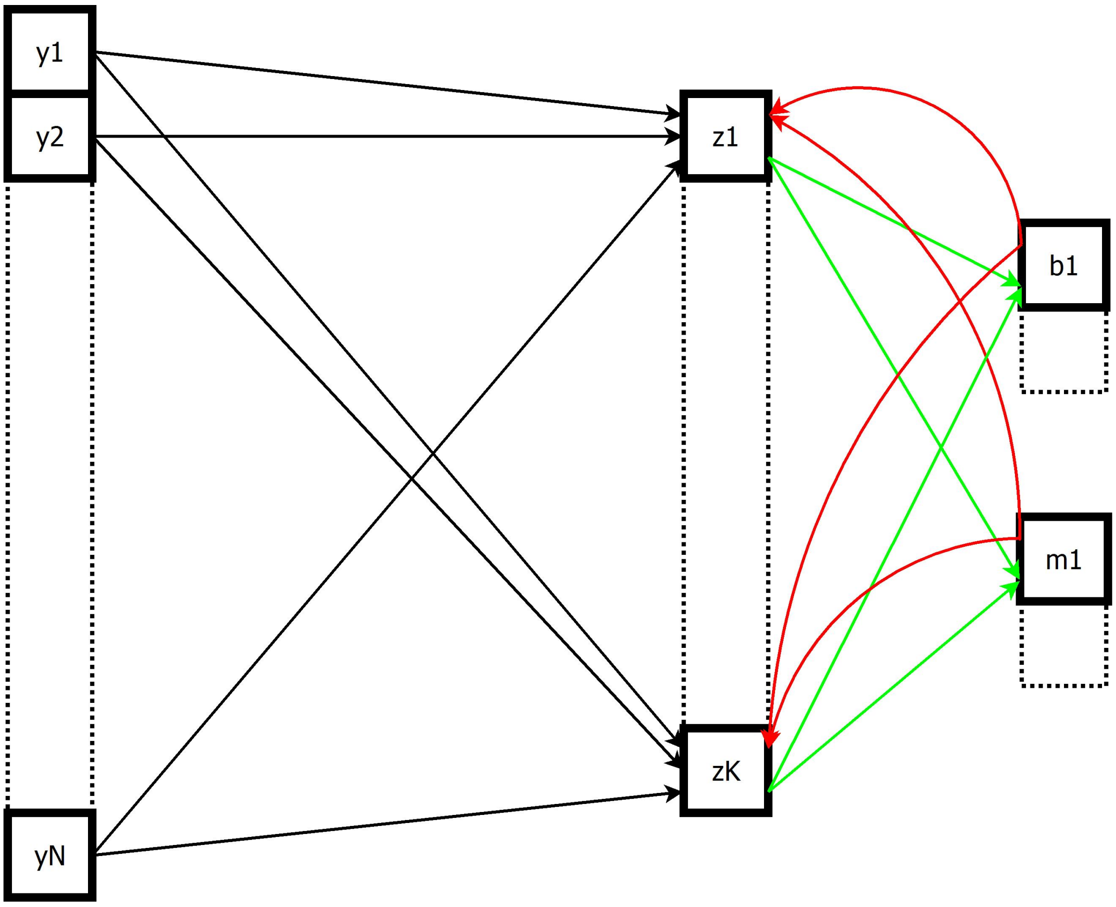

Figure 2.1: The input layer units $y_1, \ldots, y_N$ are connected to the z-layer units $z_1, \ldots, z_K$ via weighted static synaptic feed-forward connections (black). Each z-unit z_k is connected to every input unit y_n. The two populations of interneurons consist of M Martinotti Cells $m_1, \ldots, m_M$ and B Basket Cells $b_1, \ldots, b_B$, where each inhibitory unit b_b or respectively m_m is connected to all z-units and vice versa. The z-units and interneurons excite (green) or respectively inhibit (red) each other via dynamic synaptic connections (see Section 2.1.3).

current $I_{\text{exc}}(t)$ minus an inhibitory post-synaptic current $I_{\text{inh}}(t)$. A small fixed random bias w_0 modifies the excitability of each individual PC. The rate is additionally influenced by a potential U_{H} contributed by a homeostasis mechanism being described later in this section. The rate's dependency on the membrane potential is modeled as an exponential function given by

$$r_k(t) = c_{\text{rate}} \cdot e^{\frac{u_k(t)+U_{\text{H}}(t)+w_0}{\Delta_{\text{u}}}}, \tag{2.1}$$

where $c_{\text{rate}} = 10^3$ Hz is a constant factor and $\Delta_{\text{u}} = 1$ mV.
The membrane potential is given by

$$u_k(t) = u_k(t-1) \cdot e^{\frac{-dt}{\tau_{\text{m}}}} + R_{\text{m}} \cdot (I_{\text{exc}}(t) - I_{\text{inh}}(t)) \cdot (1 - e^{\frac{-dt}{\tau_{\text{m}}}}), \tag{2.2}$$

with R_m being a formal membrane resistance of 1 MΩ.
After updating membrane potential and rate at each distinct time-step every PC z_k produces a spike with a probability $p_\text{spike} = dt \cdot r_k(t)$.

2.1.2.1 Synaptic Input

Each PC z_k is connected to all N input units $y_1, \ldots, y_N$ via weighted static synapses, where w_{kn} is the weight of the synaptic connection from input y_n to z_k. The dimension of the weights is chosen to be nano-ampere (nA). The binary spike input $y_n(t)$ induces alpha-shaped post-synaptic currents. Their linear sum comprises the excitatory post-synaptic current $I_\text{exc}(t)$ being given by

$$I_\text{exc}(t) = I_\text{exc}(t-1) \cdot e^{\frac{-dt}{\tau_\text{EPSC}}} + (1 - e^{\frac{-dt}{\tau_\text{EPSC}}}) \cdot \sum_{n=1}^{N} w_{kn} y_n(t), \tag{2.3}$$

where $\tau_\text{EPSC} = 3$ ms is its time-constant of exponential decay.
The inhibitory current $I_\text{inh}(t)$ is constituted by the linear sum of inhibitory post-synaptic currents induced via dynamic synapses as described in 2.1.3. Inhibitory post-synaptic currents decay with a longer time-constant τ_IPSC of 6 ms.

2.1.2.2 Learning Mechanism

Learning is triggered by a spike of z_k at a time t_spike after a learn-lag of 1 ms. Then all synaptic weights of z_k are updated according to the learning rule

$$\Delta w_{ks} = \eta \left(c \cdot e^{-w_{ks}} \cdot y_{s\text{EPSP}}(t_\text{spike}) - 1 \right), \tag{2.4}$$

where $c = 181$ is a constant scaling factor and $y_{s\text{EPSP}}$ is the post-synaptic potential of synapse s at t_spike.
The inhibition via Martinotti Cells plays a special role in the learning process, as a spike of any MC at t_spike of z_k will intercept the respective weight update.

2.1.2.3 Homeostatic Mechanism

The role of the homeostatic mechanism implemented in z-layer units is to prevent single PCs from becoming either inactive or from firing permanently at high rates. This is performed by adding a rate-dependent homeostatic potential which either will increase or decrease the excitability of a PC if its firing rate differs from a target firing rate r^{T}. A leaky integrator is used to estimate the firing rate

$$\hat{r}_k(t) = \hat{r}_k(t-1) \cdot e^{\frac{-dt}{\tau_{\mathrm{H}}}} + z_k(t-1) \cdot (1 - e^{\frac{-dt}{\tau_{\mathrm{H}}}}), \tag{2.5}$$

where τ_{H} is the homeostasis time-constant of 1000 ms and $z_k(t-1)$ is the binary spike output of z_k.
The homeostatic potential is given by

$$U_{\mathrm{H}}(t) = U_{\mathrm{H}}(t-1) - 10^{-2} \cdot (\hat{r}_k(t) - r^{\mathrm{T}}). \tag{2.6}$$

2.1.3 Inhibitory Populations

The membrane potentials of both types of inhibitory units are updated the same way as for the PCs with the exception of their membrane time-constant τ_{mInh} being only 15 ms. Their excitatory synaptic input is received via dynamic synapses [22] being either facilitating for frequent activation in the case of Martinotti Cells or depressing in the case of Basket Cells. Therefore MCs are more efficiently excited by input of higher frequency while BCs are more efficiently activated by input of lower frequency, modeling target-cell specific short term plasticity [21], [25]. The post-synaptic current at $t = t_{\mathrm{spike}} + dt_{\mathrm{transmission}}$ resulting from a presynaptic spike of z_k at t_{spike} is given by

$$I_k(t) = (1 - e^{\frac{dt}{\tau_{\mathrm{EPSC}}}}) \cdot A \cdot o(t) \cdot R(t), \tag{2.7}$$

where $dt_{\mathrm{transmission}}$ is a synaptic transmission-delay of 1 ms, A is the amplitude of synaptic efficacy, o(t) refers to the utilization of synaptic efficacy and R(t) represents the release probability given by

$$o(t) = O + [o(t_{\mathrm{last}}) + O \cdot (1 - o(t_{\mathrm{last}}))] \cdot e^{\frac{-(t - t_{\mathrm{last}})}{\tau_{\mathrm{fac}}}}, \tag{2.8}$$

and

$$R(t) = 1 + (R(t_{\mathrm{last}}) - R(t_{\mathrm{last}}) \cdot o(t_{\mathrm{last}}) - 1) \cdot e^{\frac{-(t - t_{\mathrm{last}})}{\tau_{\mathrm{dep}}}}, \tag{2.9}$$

where t_{last} is the time of the last excitation of the respective dynamic synapse. τ_{fac} and τ_{dep} are facilitation and depression time-constants. Table 2.1 shows the respective parameters for the dynamic synapses used in the model.

type of synapse	$PC \rightarrow BC$	$PC \rightarrow MC$	$BC \rightarrow PC$	$MC \rightarrow PC$	unit
O	0.74	$1.2 \cdot 10^{-4}$	0.63	0.7	1
τ_{fac}	169	1000	1000	997	ms
τ_{dep}	221	1.3	259	241	ms

TABLE 2.1: Parameters of dynamic synapses used in the model.

The firing rate r_b for BCs and r_m for MCs is calculated differently than for PCs and is given by

$$r_b(t) = r_{\text{base}} + \frac{m}{1 + e^{\frac{x_{\text{half}} - (u_b(t) + v_{\text{Mshift}})}{\Delta v_{\text{B}}}}} \tag{2.10}$$

and by

$$r_m(t) = y_0 + A_2 \cdot e^{\frac{u_m(t) + v_{\text{Mshift}}}{\Delta v_{\text{M}}}} \tag{2.11}$$

respectively, where the parameters can be found in Table 2.2.

BC		MC	
r_{base}	-5.22 Hz	A_2	479.76 Hz
m	816.74 Hz	y_0	4.0 Hz
x_{half}	-37.419 mV	Δv_{M}	10 mV
Δv_{B}	0.89904 mV	v_{Mshift}	-42 mV
v_{Mshift}	-42 mV		

TABLE 2.2: Parameters of the rate calculation for Basket Cells and Martinotti Cells.

2.2 Model Adaptations

Inspired by research of Matthew Larkum et al. [30], [32], [33], [34] the focus of this work lies on layer 5 pyramidal cells, especially on the impact of bursts triggered by dendritic Ca^{2+} spikes on their computational power. For this purpose a simple yet plausible formal bursting mechanism was incorporated into the model.
Pyramidal neurons make up about 70 to 80 % of the cortex. Their characteristic bursts of 2 to 4 spikes with about 200 Hz are thought to be a fundamental coding mechanism [30]. An important feature of burst initiation is the mutual influence of the somatic Na^+ and the dendritic Ca^{2+} spike initiation zone via the apical dendrite. A single Na^+ action potential is able to facilitate the initiation of Ca^{2+} action potentials at the Ca^{2+} spike initiation zone near the apical tuft of layer 5 pyramidal cells. During this back

propagation activated Ca^{2+} spike firing (BAC firing) a back propagating somatic action potential decreases the threshold of the Ca^{2+} spike initiation within a time window of 3 to 7 ms after the somatic action potential [35], [33].
The burst of action potentials resulting from the plateau-type potential of a Ca^{2+} spike is thought to trigger spike-timing dependent synaptic plasticity [28], [36] as well as to link feed-forward input with feedback input [30] (which is currently not implemented in the model). Another important detail in this relation is the control of synaptic plasticity exerted by Martinotti Cells being able to suppress the initiation of dendritic Ca^{2+} spikes [19], [20].
In order to model this mechanism the currents I_{exc} and I_{inhMC} from the original point neuron model were used to calculate a virtual dendritic potential $u_{\mathrm{Burst}}(t)$. To meet the prerequisites for the incorporation of the bursting mechanism a few minor changes had to be introduced into the model being described in the following subsections.

2.2.1 Model Changes

2.2.1.1 Refractory Times

For distinguishing high-frequency bursts from regular spiking activity the frequency of regular spiking had to be limited. This was performed by introducing a refractory time t_{refrac} of 12 ms preventing the somatic spike initiation zone from firing at high rates, which occurred in the original model with frequencies of up to 1000 Hz, being the maximal possible frequency for a simulation with discrete time-steps of 1 ms. Because of this adjustment the synaptic efficacies of connections from the z-layer to the inhibitory populations had to be increased in order to provide sufficient excitation.

2.2.1.2 Prevention of negative Weights

In the original model weights of input synapses can become negative and with this induce an inhibitory current instead of excitation. To prevent this implausible behavior a hard-cap was introduced for weights becoming smaller than zero, where those weights are set to zero instead.

2.2.1.3 Control of Weight Updates

Long term plasticity is now only induced by bursts of high frequent spikes with more than 100 Hz, where regular spiking does not trigger any weight updates anymore [28].
Also the effect of inhibitory input from Martinotti Cells was changed. In the original

model MCs directly prevented synaptic plasticity by intercepting weight updates. In the current model their inhibitory currents prevent somatic action potentials from decreasing the threshold for initiation of dendritic Ca^{2+} spikes for a period of 9 ms [19], [20].

2.2.2 Bursting Mechanism

Figure 2.2 shows the schematic of the bursting mechanism. In contrast to the stochastic somatic firing mechanism bursts are triggered by $u_{\mathrm{Burst}}(t)$ crossing a dynamic threshold $u_{\mathrm{tresh}}(t)$. The virtual dendritic membrane potential $u_{\mathrm{Burst}}(t)$ depends on the excitatory feed-forward input of the input layer and the inhibition by Martinotti Cells and is given by

$$u_{\mathrm{Burst}}(t) = u_{\mathrm{Burst}}(t-1) \cdot e^{\frac{-dt}{\tau_{\mathrm{dend}}}} + (I_{\mathrm{exc}}(t) - I_{\mathrm{inhMC}}(t)) \cdot (1 - e^{\frac{-dt}{\tau_{\mathrm{dend}}}}), \tag{2.12}$$

where τ_{dend} is a dendritic membrane time-constant of 8 ms.
Simulations in NEURON [37] with a multi-compartment model of a layer 5 pyramidal cell from Stuart and Häusser [38] have shown, that the membrane time-constant becomes significantly smaller with increasing distance from the soma.
The somatic membrane potential $u_k(t)$ is calculated as described in Section 2.1.2 with $I_{\mathrm{inh}}(t) = I_{\mathrm{inhMC}}(t) + I_{\mathrm{inhBC}}(t)$ being the linear sum of the inhibitory currents of both populations of interneurons. The busting mechanism also underlies a refractory time $t_{\mathrm{refracBurst}}$ of 19 ms.

2.2.2.1 Bursting Profile

Inspired by Izhikevich et al 2003 [39] a random individual bursting profile is assigned to each z-unit at the begin of the simulation. The properties being individually assigned are the number of burst spikes (2 to 4), the frequency of the burst spikes (150 to 200 Hz) and the delay t_{decrease} until the threshold decrease of the bursting mechanism is maximal (3 to 7 ms). Each burst of a z-unit z_k is a replay of its bursting pattern. While the learning speed of units with a higher number of burst spikes is faster because more weight updates are performed, the units where t_{decrese} is lower are bursting and therefore updating their weights earlier than those with a larger threshold decrease delay.

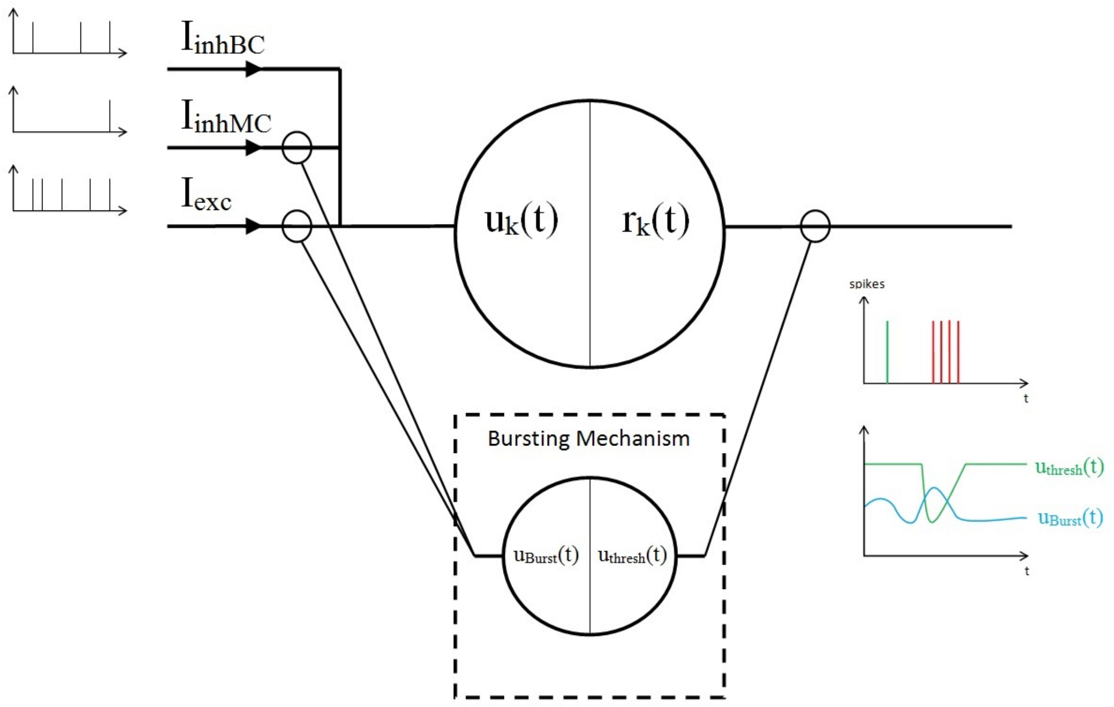

FIGURE 2.2: The illustration shows the bursting mechanism's functionality. The currents I_{exc} and I_{inhMC} of the respective z-unit z_k are used to calculate a virtual potential $u_{\text{Burst}}(t)$ which operates the threshold function of the mechanism. A precursor spike(green) of the z-unit causes the threshold $u_{\text{thresh}}(t)$ to be temporarily decreased. Continuous stimulation then is able to trigger a burst of spikes(red).

2.2.2.2 Threshold Decrease

A spike of z-unit z_k at a time t_{last} will decrease the threshold for burst initiation by a factor $d_{\max}$ of maximal 55 % [30]. The temporal decrease is modeled by two antagonistic sigmoidals and is given by

$$u_{\text{thresh}}(t) = \hat{u}_{\text{thresh}} \cdot \left[1 - d_{\max} \cdot \left(\frac{1}{1 + \exp \frac{\mu_1 + t_{\text{last}} + t_{\text{decrease}} - t}{s_1}} - \frac{1}{1 + \exp \frac{\mu_2 + t_{\text{last}} + t_{\text{decrease}} - t}{s_2}} \right) \right], \tag{2.13}$$

where $\hat{u}_{\text{thresh}}$ is the undecreased and therefore maximal value of the threshold. The remaining parameters of the function can be found in Table 2.3. An example for the time course of the threshold decrease is given in Figure 2.3.

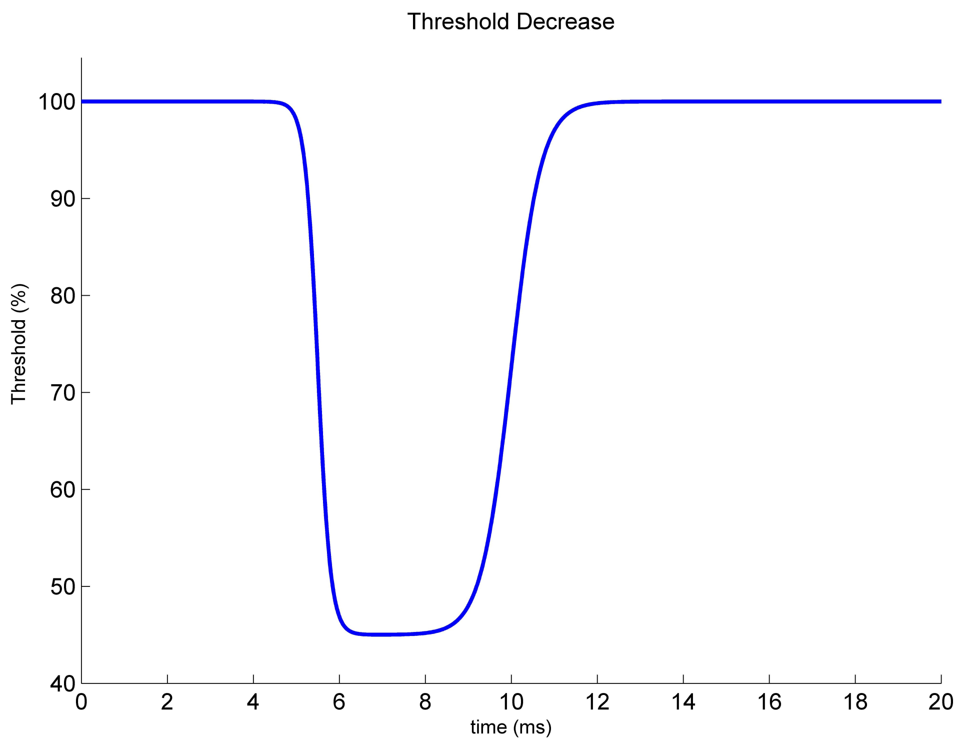

FIGURE 2.3: Time course of the threshold decrease described by Equation 2.13 as result of a spike at $t_{\text{last}} = 0$ using the parameters in table 2.3 with $t_{\text{decrease}} = 7$ ms.

μ_1	-1.5 ms
μ_2	3.0 ms
s_1	0.15 ms
s_2	0.35 ms

TABLE 2.3: Parameters of the threshold decrase function.

Chapter 3

Simulations and Results

3.1 Task 1: Learning Rate-based Patterns

This pattern recognition task is directly adopted from Stefan Häuslers original code [31]. Two classes of patterns are introduced, where the first class has twice the probability to be presented to the network than the second class. For this task ten distinct patterns are randomly generated, where five of them have a probability to be presented of $\frac{2}{15}$ and the other five have a probability of $\frac{1}{15}$. In the simulation setup a network of $N = 400$ input neurons is used with $K = 80$ pyramidal neurons in the z-layer along with ten Basket Cells and Martinotti Cells each. The simulation is performed for 1000000 time-steps of $dt = 1$ ms or respectively 1000 seconds. This task puts the network's ability of self-organization to the test, where learning the patterns with lower probabilities comprises a difficult challenge.

3.1.1 Pattern Generation

A pattern is comprised by a firing rate profile of the N input neurons, where $\frac{1}{4}$ of them is set to a *high* firing rate of 60 Hz and the rest is set to a *low* firing rate of 2 Hz representing background noise. Each pattern is generated by a random permutation of these *high* and *low* input neurons.

3.1.2 Pattern Presentation

Patterns are presented to the model in form of Poisson spike trains of 200 ms duration alternating with spacer periods of the same length, where all of the N input-neurons are set to *low*. The firing rate of *low* input neurons is constant during the presentation of a pattern with a spike probability of $dt \cdot r_n(t)$, where $r_n(t)$ refers to the rate or firing frequency of input unit y_n. The firing rate of the *high* input neurons is additionally shaped over the duration of pattern presentation and is given by

$$r_n(t) = r_{\text{peak}} \left(e^{-0.016 \cdot t} - e^{-0.024 \cdot t}\right) + 20 \text{ Hz} \left(1 - e^{-0.008 \cdot t}\right)\Big|_{t=0}^{200} \tag{3.1}$$

with

$$r_{\text{peak}} = \frac{60 \text{ Hz} - 20 \text{ Hz}}{4 \cdot \max\limits_t \left(e^{-0.016 \cdot t} - e^{-0.024 \cdot t}\right)} - 20 \text{ Hz}\Bigg|_{t=0}^{200} . \tag{3.2}$$

The sequence of patterns is rolled out for the duration of the simulation using the probabilities of the respective patterns. Figure 3.7 shows an example of the network's spike input.

3.1.3 Results

Figure 3.3 shows that the bursting model has a significantly improved pattern learning performance (see Section 4.1.2) for the task described in 3.1, and is always able to recognize all patterns with high probability and all patterns with low probability after a time of about 300 to 400 seconds. It can be seen in Figures 3.1 and 3.2 that specialization occurs faster and more directly. The reason for this relies on two major effects.
At first the bursting mechanism constitutes an efficient filter for repetitive excitation at a naive stage of learning, and afterwards selectively filters updates for the recognized excitation pattern. At this early stage a predecessor spike, decreasing the threshold of the burst initiation, will be necessary to trigger a burst in order to recognize and learn a pattern.
After some recurrences of this pattern the weights will become strong enough to trigger a burst without predecessor spike, enabling a powerful property of the circuit. Expert neurons, which already have specialized themselves for a distinct pattern, now will preempt less specialized neurons with bursting and therefore achieve a quasi-ideal inhibition. The expert's burst excites Martinotti Cells, which will in turn prevent the decrease in bursting threshold of other units what ultimately leads to prohibiting these units to learn a distinct pattern.
Ideal inhibition usually refers to inhibition in hard WTA circuits, where only a single output unit can be active at a discrete point in time. This so called winner instantly inhibits all other output units if it becomes active. In soft WTA circuits, such as the SEM model, the inhibition is at least ideal in aspect of timing. A spike of a unit instantly contributes to the inhibition signal without any delay [14]. In biological plausible models inhibition faces the problem of a temporal delay. The output of excitatory neurons has

to excite inhibitory neurons, which then will in turn provide inhibition to the excitatory cells. Because this process takes some milliseconds time (for lateral inhibition as well), ideal or at least instantaneous inhibition is obviously not possible in such circuits [10]. The property of a delayed decrease in bursting threshold now allows biological plausible inhibition to intercept burst firing of output units in time.
Altogether this behavior enables the z-units to distribute themselves evenly over the patterns and the network will converge fast and efficiently as can be seen in Figure 3.4. The analysis of update qualities (see Section 4.1.4) in Figures 3.5 and 3.6 shows the filter effect of the bursting mechanism for finally learned patterns. While there is little difference in update quality for currently presented patterns in both models, the update qualities for finally learned patterns differs considerably. The scope of the variance in this plot is much narrower for the update qualities of the bursting model. This shows that the z-units of the bursting model update more selectively for a certain pattern than the original model's z-units do. Figure 3.7 shows the spiking and bursting behavior of the model at the start and at the end of the learning process. After learning the bursts indicating recognition of a certain pattern can be clearly distinguished from the unsystematic spiking activity at an early stage of learning. Also the network output becomes quite sparse.

3.2 Task 2: Feature Learning with crossing Bars

In this task the 400 input neurons are considered to be arranged as a 20×20 matrix and the patterns are comprised by non-overlapping variants of vertical and horizontal bars of neurons with *high* firing rate with a width of three columns or lines respectively. These patterns are presented to the network as a random combination of a vertical and a horizontal bar resulting in 111 *high* and 289 *low* neurons. The frequencies were adjusted by the factors $\frac{100}{111}$ and $\frac{300}{289}$ to obtain approximately the same expectation for the number of input spikes during a pattern presentation as in Task 1. A small error remains from the discretization of the time space or non-infinitesimal bin-size dt. Since there are six possible positions for each type of bar, the number of distinct combinations of a vertical and a horizontal bar is 36. To avoid over-fitting the number of z-neurons was reduced to 24 for this task while the remaining setup remained the same as for Task 1.
This task is an implementation of the Bars-Problem (or Bars-Test) introduced by Peter Földiák in 1990 [40], which has become a standard problem for unsupervised learning [41]. The goal of this task is to put the network's ability of internal knowledge representation to the test. There are several ways how a network can derive and store information about patterns and hidden causes. A sparse representation reduces the amount of elements while preserving the information about a hidden cause. With the

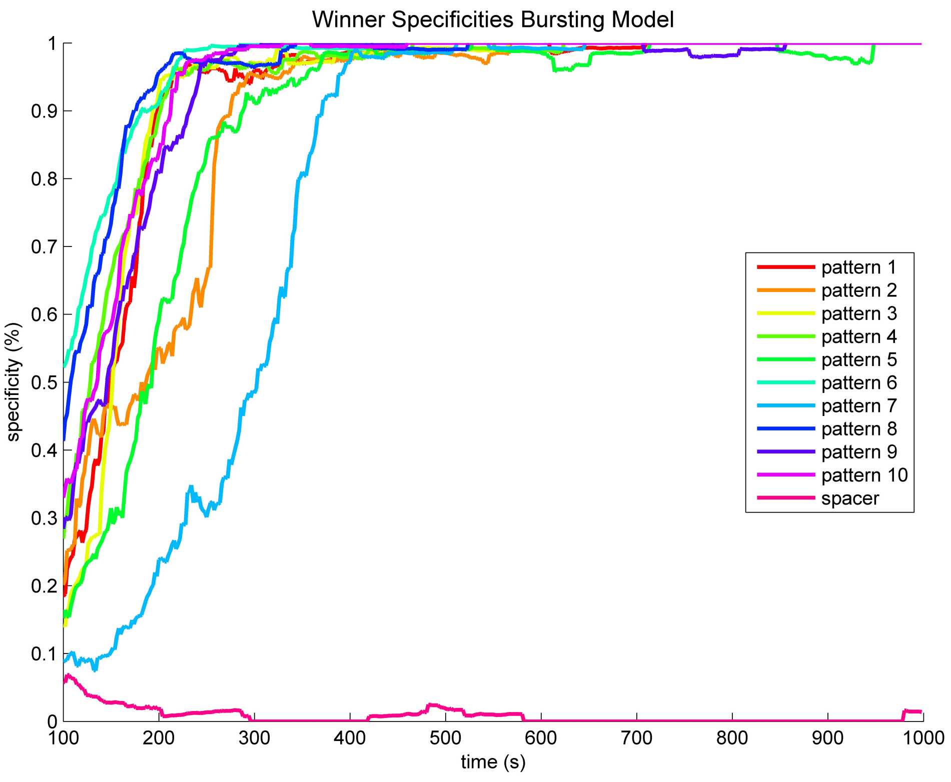

Figure 3.1: Temporal development of the pattern specificity of the winner neurons from the bursting model.

reduction of redundancy of information also generalization is promoted.
A distinction is drawn between a local representation and a distributed representation. In local representations a single unit stores all information about a pattern, while in distributed representations common statistical regularities or features of a pattern are learned by a set of units, each representing a distinct feature. Obviously the representational capacity of distributed representations is higher. For local representations the number of learnable patterns is equal to the number of z-units. In a distributed representation each unit learns a certain feature and the number of representable patterns is the number of possible combinations of these features.
Regarding the task described in this section the information can be represented in three different ways. In the first, being the most straight forward but undesired version, each unit learns a single combination of a horizontal and a vertical bar, depicting a local representation. A sparse local representation would be to learn the intersection of both bars, being a characteristic feature of each combination of bars. The fourth panel in the first row in Figure 3.9 illustrates such a representation. In a distributed representation a unit learns a feature in form of a single horizontal or vertical bar. This way only twelve units would be necessary to represent all 36 possible combinations of a horizontal and a

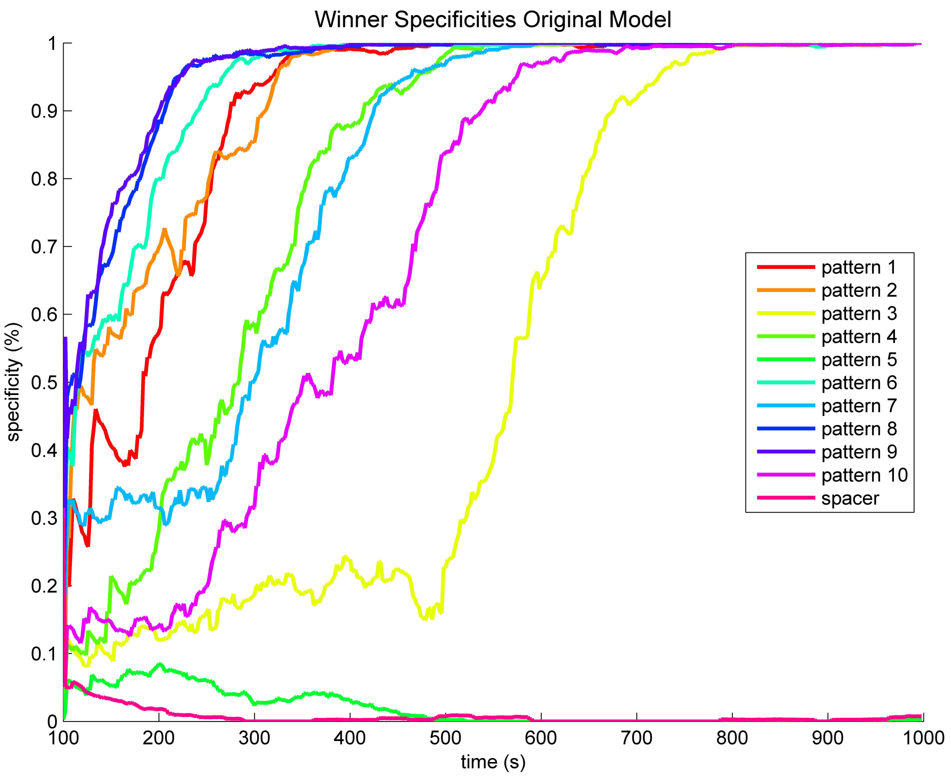

FIGURE 3.2: Temporal development of the pattern specificity of the winner neurons from the original model.

vertical bar. In order to promote feature learning in form of single bars instead of two crossing bars a different learning rule [42] was used for this task instead of Equation 2.4, being given by

$$\Delta w_{ks} = \eta \left[c \cdot \left(1 + \frac{1}{(a \cdot w_{ks} + b)^2} \right) y_{s\text{EPSP}} - 1 \right] . \tag{3.3}$$

Equation 3.3 results from an approximation in consideration of biological plausibility of a learning rule performing gradient ascent on a log-likelihood $\tilde{p}(\mathbf{y}|\mathbf{z}, \mathbf{W})$ (see Section 1.3.4) in the context of a soft winner-take-all circuit.
In order to display a z-unit's knowledge about the input patterns the weights of the z-units are arranged the same way as the input-units, being visualized in form of a greyscale map, where black indicates a strong weight and white indicates a weight being zero or very weak.

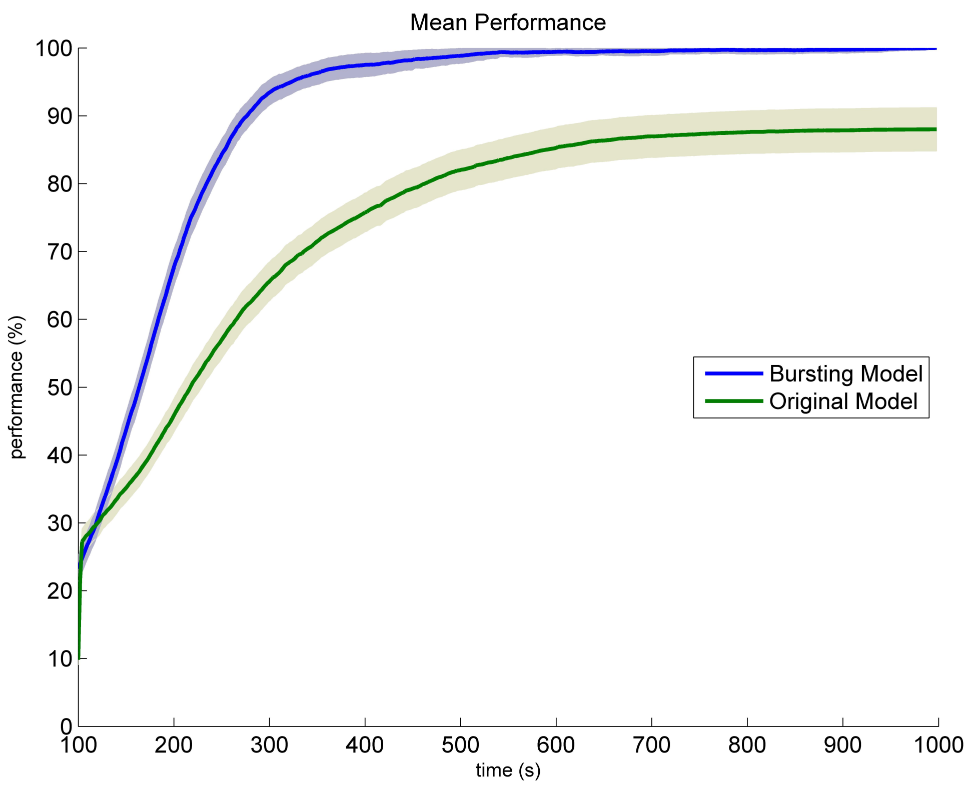

FIGURE 3.3: Temporal development of the mean performance and variance of original and bursting model for 15 simulation runs. The z-units of the bursting model are able to distribute themselves better over the learnable patterns achieving a maximal performance.

3.2.1 Results

Figures 3.8 and 3.9 show the direct comparison of the weights learned by the model using either the SEM learning rule or Equation 3.3. With the SEM learning rule a z-unit is in most cases unable to decide for a distinct bar. What happens in detail is the following: When a z-unit updates its weights for a pattern the first time both bars are learned with about equal strength. If in the next step a different pattern triggers a weight update, one bar (bar A) is most likely a common feature shared with the first pattern, while the other bar (bar B) is new to the unit. With this the weights for bar A are strengthened, but the strength is increased less than in the first update. The information of the new bar (bar C) is added to the units' weights and the information for bar B fades. In the next step the problem with the SEM learning rule comes into effect. If an update occurs during a pattern presentation where bar A is not present, but bar B or C, the weights for bar A are much more decreased for bar A than they were increased in the previous step, while bar B and C are strengthened or decreased respectively.

Because the change of gradient in weight update strength is very steep this leads to weak

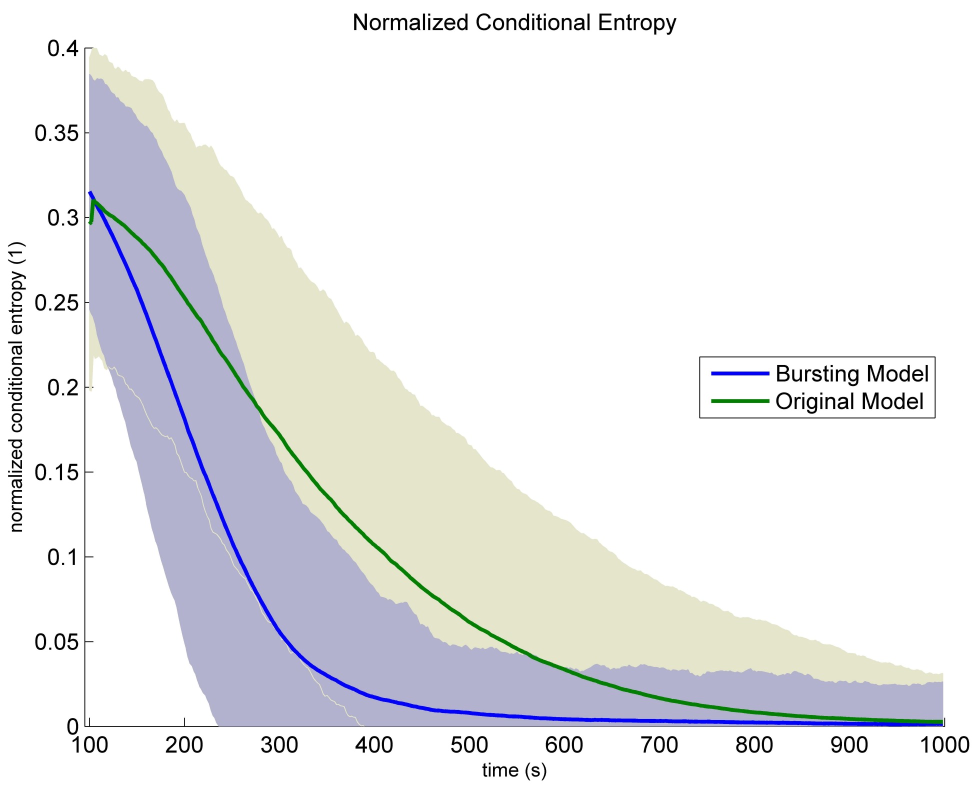

FIGURE 3.4: Temporal development of the mean network entropy and variance of original and bursting model for 15 simulation runs. The bursting model converges faster due to more selective weight updates (see Figure 3.5).

updates for reoccurring bars and strong updates for the bar of orthogonal orientation to the reoccurring bar. Additionally the weights for bars with strong weights are decreased largely if an update occurs for a pattern where this bar is not present. Ultimately the model is mostly unable to average-out single bars or decide for a single bar, while the learning rule from Equation 3.3 supports averaging-out of sparse features and therefore nearly all bars can be distinguished after learning.

The results for this task show that a different learning rule than the SEM learning rule is necessary for extraction of common features from patterns. While Equation 3.3 constitutes a relatively simple mechanism which is able to achieve this goal, a more sophisticated model of synaptic plasticity might be necessary to match biology. Jochen Triesch illustrated in his work from 2007 [41] the impact of different mechanisms of synaptic plasticity in a model where intrinsic plasticity (IP) was combined with Hebbian learning. Proposing that synaptic plasticity is arising from the computational goal of sparse signaling, he and Daniel Krieg introduced a model where the spike-timing dependent plasticity (STDP) mechanism actively maximizes the sparseness of the distribution of synaptic efficacies [43].

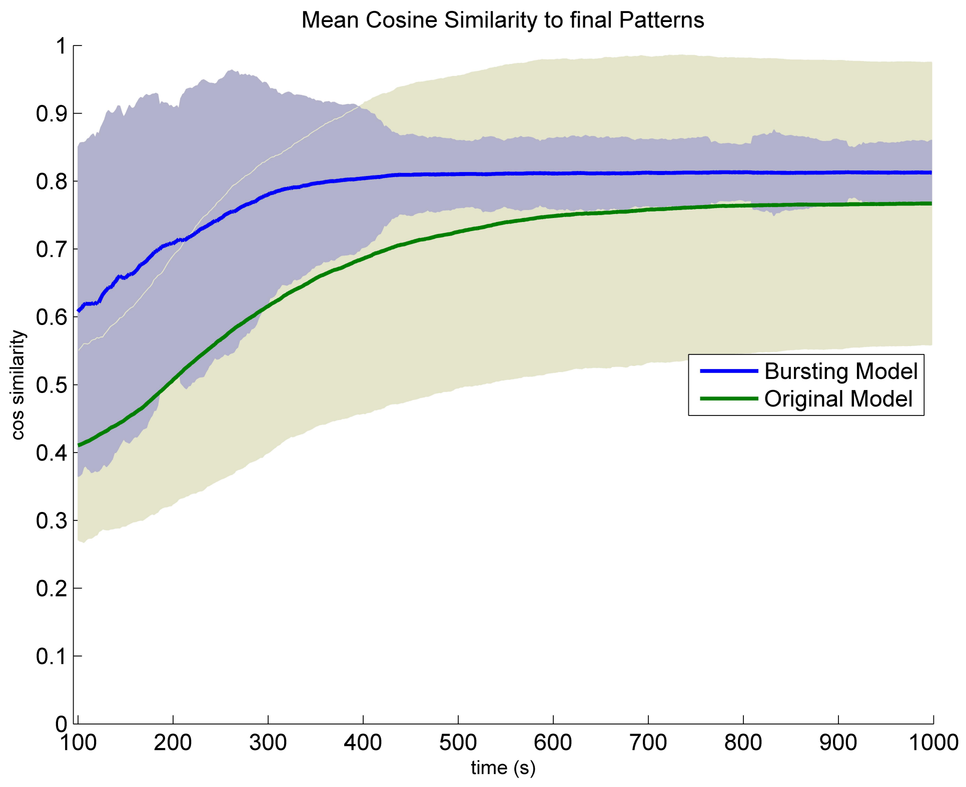

Figure 3.5: Temporal development of the mean cosine similarity of the weight updates of the winner neurons to their final patterns of original and bursting model for 15 simulation runs. The winners of the bursting model specialize faster and update their weights more selectively for their finally learned pattern.

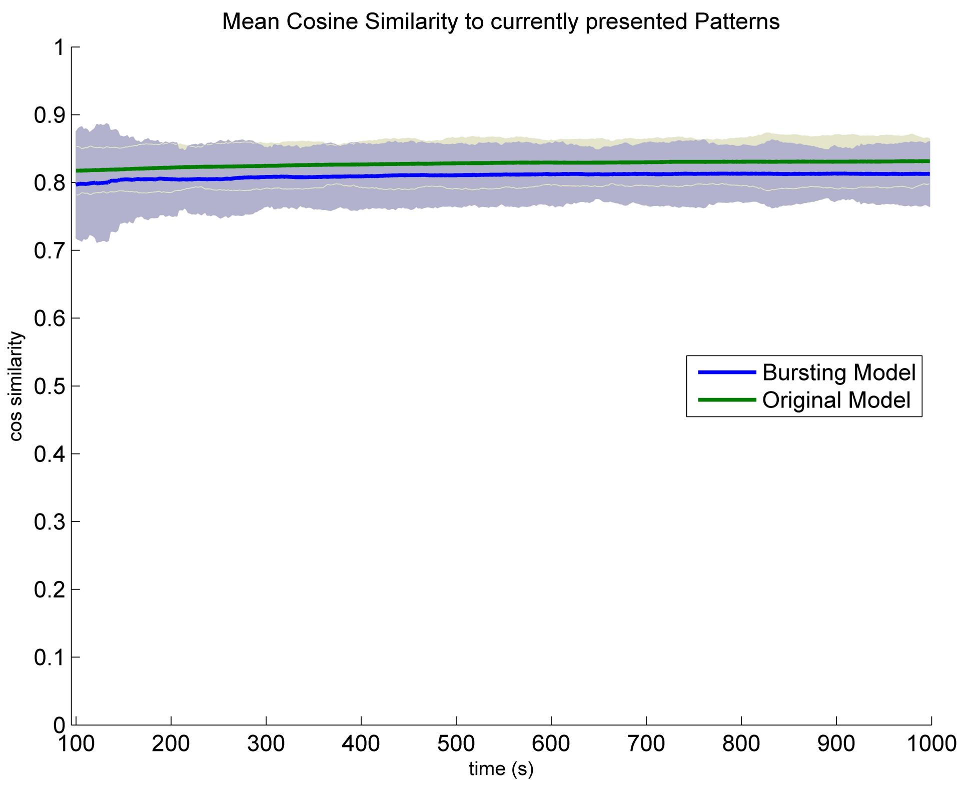

FIGURE 3.6: Temporal development of the mean cosine similarity of the weight updates of the winner neurons to currently presented patterns of original and bursting model for 15 simulation runs. There is no significant difference between the models.

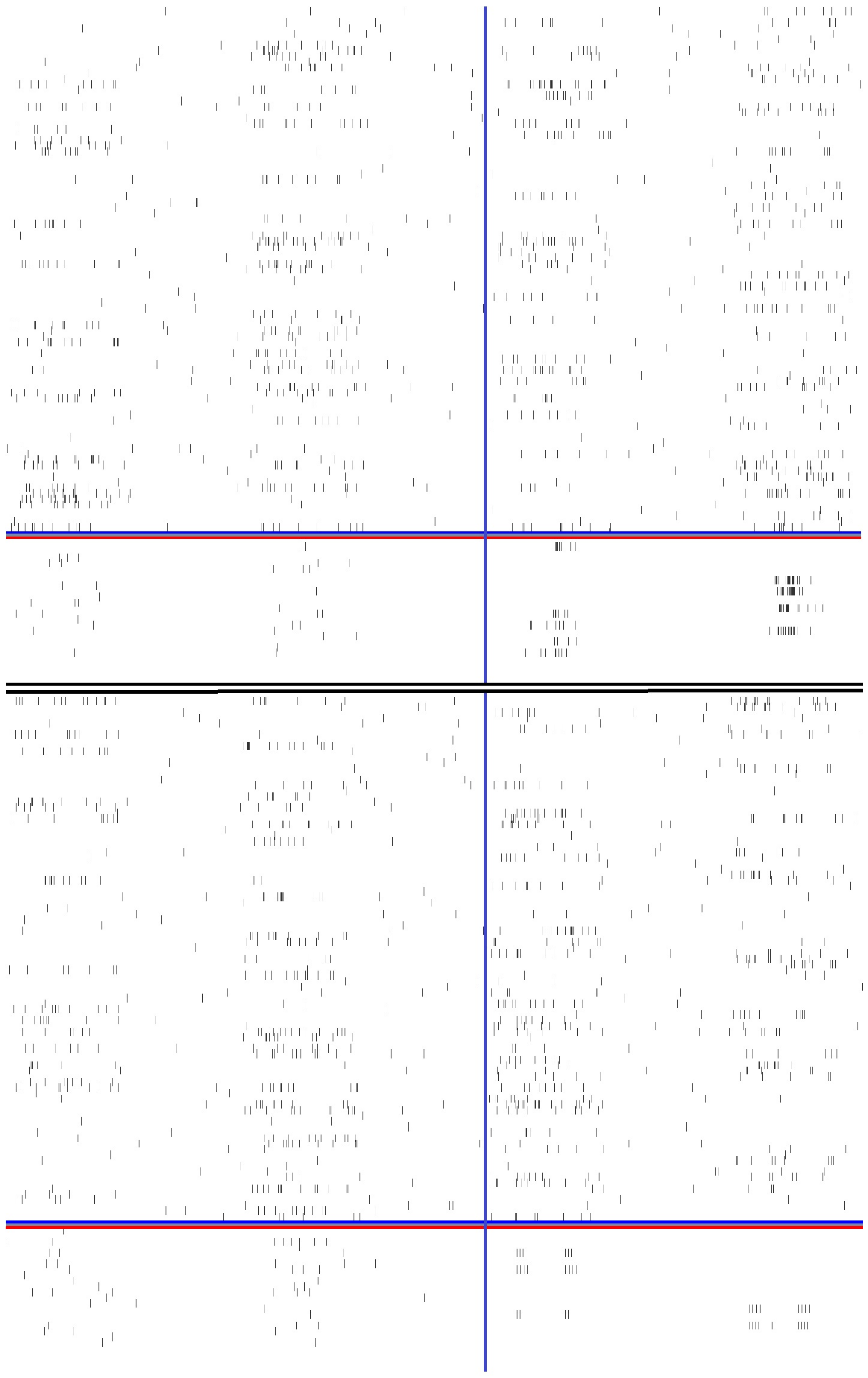

Figure 3.7: Comparison of the spike input(large bands) and output(small bands) of the original model (above) and the bursting model (below). While after learning (right hand side) the original model's z-units fire at high frequencies for distinct patterns, the z-units of the bursting model burst selectively for distinct patterns.

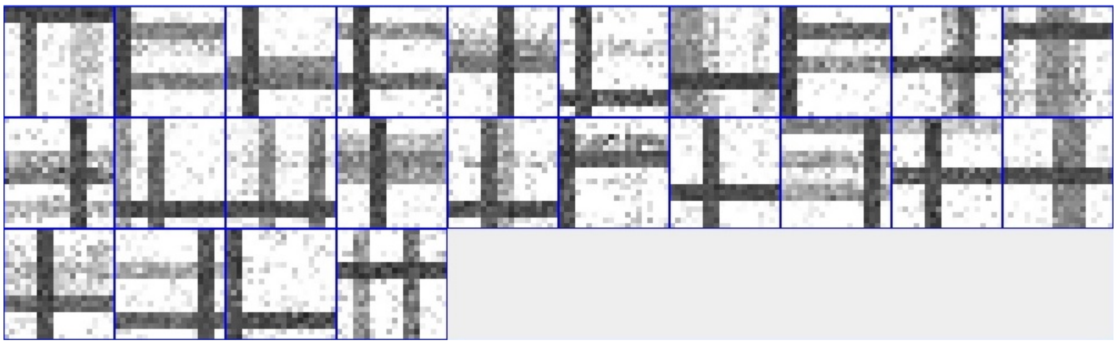

FIGURE 3.8: Weights of the z-units after learning using the SEM learning rule. The model is mostly unable to distinguish distinct bars.

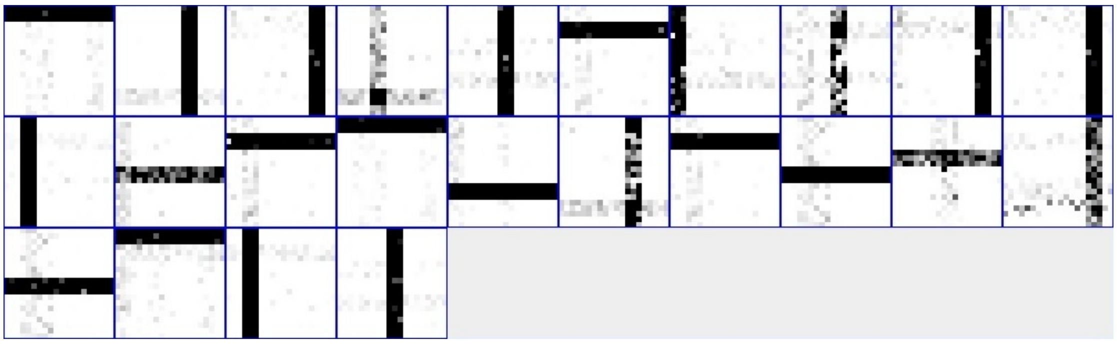

FIGURE 3.9: Weights of the z-units after learning. Nearly all distinct bars were learned by the model using the learning rule from Equation 3.3.

Chapter 4

Methods

4.1 Measuring Model Performance

To display learning performance as well as its temporal development the evaluation of the model is implemented as sliding window over the output of the z-layer. The size of the sliding window was chosen to cover 10 % of the simulation's time, propagating with a step-width of 10 times the duration of a pattern presentation. For the task described in 3.1 the sliding window has the size of 100 seconds and propagates with a step width of 2 seconds.

4.1.1 Specificity and Winner Selection

To measure how well a z-unit z_k has captured a certain pattern the pattern specificity $S_k(p)$ of its firing behavior is calculated as number of spikes $\#_{\text{spikes}}(z_k, p)$ it has fired for pattern p in relation to all spikes $\#_{\text{spikes}}(z_k)$ from z_k within the sliding window. This is equivalent to the posterior probability for pattern p being presented given z_k is spiking. The spacer in between pattern presentations is also treated as a distinct pattern.

$$S_k(p) = \frac{\#_{\text{spikes}}(z_k, p)}{\#_{\text{spikes}}(z_k)} = p(P = p|z_k) \tag{4.1}$$

After evaluating the specificities for all patterns of every z-neuron for all sliding window positions a winner for every pattern is selected, being the neuron with the highest specificity for this pattern at the last sliding window position.

4.1.2 Rating Learning Performance

To quantify how well the model has completed the learning task a performance measure was introduced, where every pattern excluding the spacer contributes the same amount to the total performance. The performance P is therefore defined as the sum of the specificities of all winners z_p for their pattern p in relation to the number of patterns.

$$P = \frac{\sum\limits_{p \in P} S(z_p, p)}{|P|} \tag{4.2}$$

Obtaining a specificity of 1 for every winner will therefore result in a performance of 100 %. A separate measure for sensitivity is not necessarily needed, since the homeostasis mechanism described in 2.1.2.3 would always cause the z-units to spike unspecifically if they do not spike or burst sufficiently for their respective pattern. This leads to an increase in spiking activity for their pattern coming with increasing specificity. Z-units with a high specificity for their pattern will also show a high sensitivity for this pattern.

4.1.3 Conditional Entropy

Another measure for the convergence speed of the learning process is the decrease in entropy of the network output given a hidden cause over time.
Similarly to Nessler et al. 2013 [14] the temporal development of the normalized conditional entropy $H(P|Z)/H(P,Z)$ was computed using again the sliding window as already described. Since the model has mostly sparse spiking activity only spiking information was regarded, discarding all non-spiking times. The conditional entropy $H(P|Z)$ and the joint entropy $H(P,Z)$ are given by

$$H(P|Z) = \sum_{k=1}^{K} p(z_k) \sum_{p \in P} -p(P = p|z_k) \log_2 p(P = p|z_k) \tag{4.3}$$

and

$$H(P,Z) = \sum_{k=1}^{K} \sum_{p \in P} -p(z_k) p(P = p|z_k) \log_2 p(z_k) p(P = p|z_k), \tag{4.4}$$

with $p(z_k)$ given by

$$p(z_k) = \frac{\#_{\text{spikes}}(z_k)}{\sum\limits_{k=1}^{K} \#_{\text{spikes}}(z_k)}. \tag{4.5}$$

4.1.4 Quality of Weight Updates

To understand how a model is learning different patterns it is necessary to have insight into the way neurons update their weights. To quantify the consensus between patterns and weight updates the cosine similarity

$$\cos\phi = \frac{\vec{p} \cdot \vec{y_t}}{\|\vec{p}\|\|\vec{y_t}\|} \tag{4.6}$$

was computed, with $\vec{p}$ consisting of the frequencies of the input Poisson spike trains and $\vec{y_t}$ being the EPSPs at the time the weights of the neuron were updated. Since all vector elements are always positive numbers, the resulting cosine similarities reside between 0 and 1.

For this evaluation only the winner neurons z_p for every pattern were regarded. The mean update quality over the winner neurons was calculated from their individual mean update qualities within the sliding window discarding winners not updating their weights in this span of time. To display the course of specialization of the network the cosine similarity was calculated for the finally learned patterns of the winner neurons as well as for the patterns being currently presented to the network at the times of the weight updates.

Chapter 5

Discussion

The analysis of the results in Chapter 3 gives rise to the questions how much insight in neuronal information processing might be provided by this modeling approach, or respectively whether these results can somehow be applied to self organization in biological neocortical neuronal circuits. It may at least be said that dendritic calcium spikes in combination with a temporary threshold decrease triggered by back propagating action potentials might contribute a significant amount to the computational power of those circuits. Still many insights on properties of these circuits and their information processing units are more or less intensively debated making argumentation on the base of computational models an even more delicate matter, considering many biological details being most likely still unknown.

A central point of interest is the interpretation of the inhibition mechanism that is comprised by the prevention of burst threshold decrease exerted by Martinotti Cells. In context of the winner-take-all circuit comprised by this work's model the delayed threshold decrease provides a time window for efficient exertion of inhibition in between pyramidal cells via excitation of Martinotti cells. Something like this might actually happen in biological circuits. Considering the main feedforward input of the PC arriving at proximal dendritic segments, while feedback input arrives at distal dendritic segments in the tuft, this mechanism might provide an interesting possibility in controlling the linkage of these two types of input. A spike triggered by the feedforward input would make the respective PC susceptible for appropriate feedback input by depolarizing the apical dendritic trunk and decreasing the burst threshold for a certain time window. If a PC already has performed the linkage it would prevent others from doing so via local inhibitory MCs. This proposes a "suggest and confirm" mechanism constituted by PCs, suggesting the outcome of a decision process by a spike response on feedforward input subsequently confirming it if the feedback from other areas suits this suggestion.

The delayed threshold decrease itself is already intriguing because of obviously not resulting from supralinear summation of bAPs and dendritic excitation since occurring a few milliseconds after arrival of the respective bAP. Eventually an enzymatic process is responsible for the delayed temporary effect on the calcium spike initiation zone. Such mechanisms are unlikely to evolve incidentally without fulfilling an advantageous function for the respective organism, therefore this detail most likely plays a distinct role in the computations performed by pyramidal cells, where the results in Section 3.1 support this estimation. It is unclear if this idea will find agreement in biological neurosiciences, but could at least inspire future modeling approaches or experiments.

5.1 Related and Future Work

The already discussed threshold decrease mechanism for calcium spikes in combination with inhibition via interneurons comprises a unique feature of this work's model distinguishing it from related work in the context of layer 5 pyramidal neurons. There are several types of modeling approaches being closely related. While models performing spike based expectation maximization were already introduced in Section 1.3 other models focus on prediction of spike-timings in electro physiological recordings by modeling a more complex dendrite including different types of nonlinearities such as NMDA spikes [44]. Others focus on more complex models of STDP like the work of Jochen Triesch [41] and Daniel Krieg [43] mentioned in Section 3.2. Also new insight on STDP on molecular level [45] might provide inspiration for new modeling approaches.
A promising next step for extending this work's model would be the introduction of a complex dendrite providing the dendritic potential for the bursting mechanism (see Section 2.2.2) as well as a dendritic signal transmission function. The three layer model introduced in Section 1.4.3 and shown in Figure 1.1 would serve as a good starting point for this task. The incorporation of NMDA spikes into the model would require to distribute dendrites on a set of dendritic branches, because coincident synaptic inputs cause NMDA spikes only if the respective synapses are located on the same dendritic branch within a vicinity of about 100 μm [27], [29]. Therefore also a continuous synaptic regrouping mechanism might be necessary to enable coupling of corresponding synapses. A difficult challenge might be providing suitable feedback input for the respective feed-forward input. This may be realized by extending the network by an additional circuit providing the feedback input based on the output of the winner-take-all circuit. Such an approach could model the linkage between feedback and feedforward input which is thought to occur at the level of neocortical layer 5 PCs. Also a more detailed modeling of interneurons would be a possible approach for a future refinement of this model.

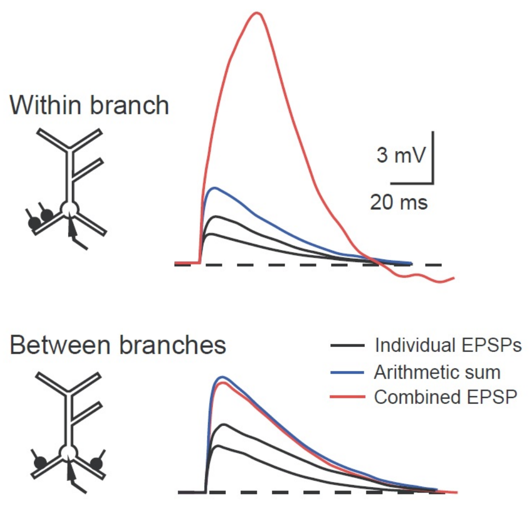

FIGURE 5.1: This illustration from [29] shows the supralinear boosting of coincident synaptic responses in form of NMDA spikes. While synapses located at the same dendritic branch are able to trigger NMDA spikes the responses are more or less linearly summed between different branches.

5.2 Conclusion

The results of this work show that there is a surprising increase of performance in self organization and pattern recognition tasks when extending the point neuron model by a burst mechanism. This result is quite robust in respect to modeling parameters as long as the bursting threshold is set in a reasonable fashion. A dynamic bursting threshold dependent on bAPs and inhibition by MCs seemingly overcomes the problem of precisely timed inhibition in circuits where inhibition occurs via spiking interneurons. It is also shown that the model of STDP or applied learning rule determines how models internally derive and represent knowledge about the distribution of their input. The large impact of small details on computational power and behavior suggest that computation at the level of individual cells might be more sophisticated and powerful than previously assumed. Eukaryotic cells are on a wholly different level of complexity than bacterial cells, which already prove to be difficult to understand. It's most likely that there are many still unknown details playing a significant role in neuronal information processing. Improvement in biological neurosciences and molecularbiological methods will lead to new insights in the functioning of neurons and synapses. Modeling approaches based on new findings and details might give rise to a new generation of computational models unleashing the power of neocortical information processing.

Appendix A

Implementation Details

A.1 General Information

The implementation of the WTA network simulation framework is comprised by two project-folders containing the models described in Chapter 2. The first is a functionally identical reimplementation of Stefan Häusler's model (see Section 2.1) based on his MATLAB code, while the second contains the adapted model described in Section 2.2. The reimplementation was realized as object oriented approach in Java$^{\mathrm{TM}}$. Purpose of this was to provide a fast simulation framework, appropriate for execution on a personal computer or notebook on the one hand, while on the other hand the modular organization allows an easy access to the model's components for extension and remodeling.

A.2 Running the Simulation

The simulation can be started by execution of the `main`-method of the Network class. The Java VM used in this work was the Java HotSpot$^{\mathrm{TM}}$ 64-Bit Server VM 24.45-b08. It is recommended to set the VM arguments `-Xmx` and `-Xms` for maximum and mimimum heap size in order to prevent the simulation from either running out of memory or becoming very slow. If the heap usage gets close to the maximum heap size, Java will make excessive use of the garbage collection, which will tremendously slow down the application.

A.3 Package Overview

Figure A.1 gives an overview on packages and classes in the framework. The following subsections will describe all functional components organized within the packages. For running different simulations the Network class will be the main address to look into. In order to remodel network units all classes representing those can be found in the `wta.cell` package. The learning tasks described in Chapter 3 are defined in the classes within the `wta.input` package. Tools for visualization and evaluation can be found in the packages `wta.gui` and `wta.evaluation`.

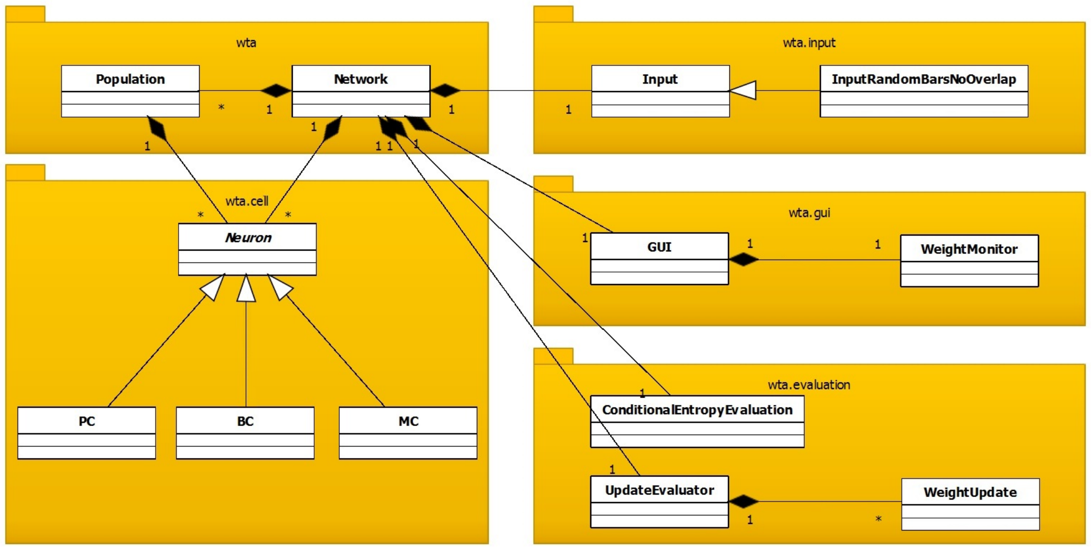

FIGURE A.1: The class diagram gives an overview on the organization of the simulation framework. Central element of the implementation is the `Network` class, containing all components. The packages organize the classes in respect to functionality. Network elements representing different types of neurons can be found in the `wta.cell` package. The other packages provide classes for generation of the network's input, visualization of spike trains and weight settings, and for evaluation of simulations.

A.3.1 Package `wta`

A.3.1.1 Class `Network`

As already mentioned the `Network` class is the heart and center of the application. The class represents and simulates the network being also responsible for organization of all of its network components and additional components.
All parameters concerning the configuration of the network can be found in the head of the source file, such as the number of z-units, number of Martinotti and Basket Cells, duration of simulation, parameters characterizing the spike input, synaptic efficacies and

many more. By evoking the `main`-method one instance of a network is created, initialized and simulated. If executed in batch-mode by setting the number of trials in the main-method, the main method will sequentially create and simulate networks, saving the simulation results for each trial. In batch-mode no graphical interface is opened to visualize the network's input and output spikes or learning process.
In the initialization process all cellular components of the network are instantiated, initialized and then assigned to populations being special objects implementing the `Callable` interface. By doing so, the units are distributed on different threads with the benefit of splitting the computational overhead on the cores of the CPU.
The simulation process is comprised by a loop feeding spike input batch-wise to the network. This was designed to enable online generation of input in order to save memory in case there is no need of saving the spike input. Since the spike input is saved in the current implementation, the framework does not make use of this feature. But it can simply be utilized by implementing a class of input not storing the created spike trains. The simulation itself is basically performed by simply updating all neurons of the network every time-step by calling their `update()` method. This happens by invoking all population-threads using an `ExecutorService`.
After simulation, the network output is evaluated and results are stored in text files.

A.3.1.2 Class `Population`

The `Population` class acts as container for neurons and has the function to divide the computational load over threads, where each population object comprises a distinct thread. On instantiation the population receives an `ArrayList` of neurons as parameter. It's recommended to assign equal numbers of each type of cell to every population, since the most time-consuming thread will become the bottleneck of computation.
This class implements the `Callable` interface comprising a specialized type of thread being able to deliver a return value on call. The thread is defined by implementing the mandatory `call()` method. In this case the thread simply prompts every neuron in the population to update its synapses, membrane potential and spike output, returning a `Boolean` object when finished. The callables are invoked simultaneously by an `ExecutorService`, where a `Future` object is created for every thread's return value. This "future" is realized the moment the thread terminates, delivering its return value. If all callables are finished or terminated otherwise, the execution of the code where the callables are invoked continues. In case of the network's simulation all populations are called in every time-step, where proceeding to the next time-step occurs when all populations are finished updating their neurons.

A.3.2 Package `wta.cell`

Information processing network units representing distinct types of neurons are combined in this package, being all derived from the abstract mother class `Neuron`. This design allows the simulation framework to treat all units the same way. The modeling of information processing in this framework can be considered being divided into three layers. While the first layer is the network level, the second level is comprised by the cell types modeled in this package. The third layer is the level of synapses, where each cell type describes its own types of synapses in the form of private classes being only accessible to themselves.

A.3.2.1 Class `Neuron`

This abstract class serves as the main interface between the simulation environment and its compartments. It dictates all derived subclasses to implement the `update()` method being called by the simulator. In this method every derived subclass has to define its specific behavior. Furthermore the `Neuron` class contains all constant parameters being common to all subclasses of neurons, such as time-constants for excitatory and inhibitory postsynaptic currents. It also serves as interface for referencing the output space for distinct types of neurons.
The third major function of this class is acting as the global clock of the simulation, storing the current time-step accessible to all subclasses. After all neurons are updated the state of the network is set to the next step by calling the static `nextTimeStep()` method of this class.

A.3.2.2 Class `PC`

The main type of information processing unit participating in the winner-take-all circuit is the z-unit representing a pyramidal cell. Constant parameters such as membrane time-constant, parameters concerning STDP and all other cell type specific parameters can be found in the head of the source code of this class.
The way PCs process their spike input is defined by the private class `InputSynapse`. Two additional types of synapses are responsible for input from Basket Cells and Martinotti Cells. Each PC instance owns a collection of each type of these synapses, connecting it to all types of inputs. The behavior of the cell is defined in its `update()` method. The cell updates its membrane potential by first updating all synapses, each responding to presynaptic spikes at the current time-step. After updating all postsynaptic currents the membrane potential is actualized with subsequent update of the output rate. In a next

step it is decided if the cell switches into burst mode (see Section 2.2) or continues in normal spiking mode. In each mode the spike output of the respective z-unit is written in its respective output space. While in burst mode the cell replays its specific bursting pattern, in normal mode output spikes are randomly generated based on the current value of the rate variable. Depending on the model (original or adapted bursting model) the synaptic weights of input synapses are ultimately updated, either when a spike is fired or when spiking in burst-mode. This is performed by calling the `updateWeight()` method of every input synapse. A boolean constant `useLR2` decides if the learning rule of Equation 2.4 is applied or Equation 3.3.

A.3.2.3 Classes `BC` and `MC`

Analogue to the `PC` class the behavior of BCs and MCs is defined in their `update()` method, while cell type specific parameters can be found in the heads of the respective source file. Both classes own just one type of private synapse class being called `PCSynapse` in both cases. Although being implemented similarly these synapse classes contain different parameter settings defining their specific short term plasticity (see Sections 1.4.2 and 2.1.3).

A.3.3 Package `wta.input`

The classes contained in this package implement the tasks introduced in Chapter 3, providing functionality for the creation of patterns, sequence of pattern presentations and the corresponding input spike trains.

A.3.3.1 Class `Input`

Representing the network input this class implements the task described in Section 3.1. By instantiating it the input is created in three steps. First the patterns are randomly generated through permutation of input units with low and high firing rate plus a pattern representing the spacer in between pattern presentations. Next the sequence of pattern presentations is randomly generated using the probabilities of the two classes of patterns to be presented. Finally the spike trains are generated and stored, organized in batches with the length of a pattern presentation. By calling the method `getBatch()` such a batch of input spikes is returned by an `Input` object, internally storing the information which batches were already delivered. Therefore repetitively calling the method will sequentially return all batches of input spike trains until the end of simulation is reached.

A.3.3.2 Classes `InputBars`, `InputRandomBarsNoOverlap`

These classes derive from the `Input` class, overriding the procedure of pattern generation, where `InputRandomBarsNoOverlap` implements the task described in Section 3.2. Note that `InputRandomBarsNoOverlap` derives from `InputRamdomBars`. Both classes generate patterns in form of a crossing horizontal bar and a vertical bar. Patterns are neither stored nor follow a sequence but rather are random combinations of bars, where in `InputRandomBarsNoOverlap` no bars of the same orientation overlap with each other resulting in fewer possible bars of the same orientation than in `InputRandomBars`.
`InputBars` generates analogously to class `Input` a fixed number of distinct patterns and a sequence of pattern presentations but in the form of crossing bars.

A.3.4 Package `wta.gui`

The graphical interface of the framework is realized by using the Java Swing toolkit. While basically the `Network` class provides the main window in form of a `JFrame`, the classes implementing visualization are each realized as `JInternalFrame` being displayed within the main `JFrame`.

A.3.4.1 Class `GUI`

Using different panels this class displays the whole spike input and output of all neurons in the network after the simulation is finished. A panel for each type of unit illustrates the spikes over the whole duration of the simulation as can be seen in Figure A.2. The borders of the panels have different colors, being blue for the input, red for z-units, green for Basket Cells and yellow for Martinotti Cells. Each Panel has a horizontal scrolling bar to scroll through the course of the simulation.

A.3.4.2 Class `WeightMonitor`

In contrast to the spike trains, being displayed at the end of the simulation, the weights of the z-units can be monitored online while the simulation is carried out. The weights are arranged in squares representing all weights of a z-unit as gray-scale pixels, where dark pixels represent large weights and bright pixels represent low weights. The appearance of the `WeightMonitor` window is shown in Figure A.3. In order to update the displayed information the `update()` method has to be called during the simulation. The current implementation of the `Network` class updates the display every ten batches of input or respectively every two seconds of simulation time.

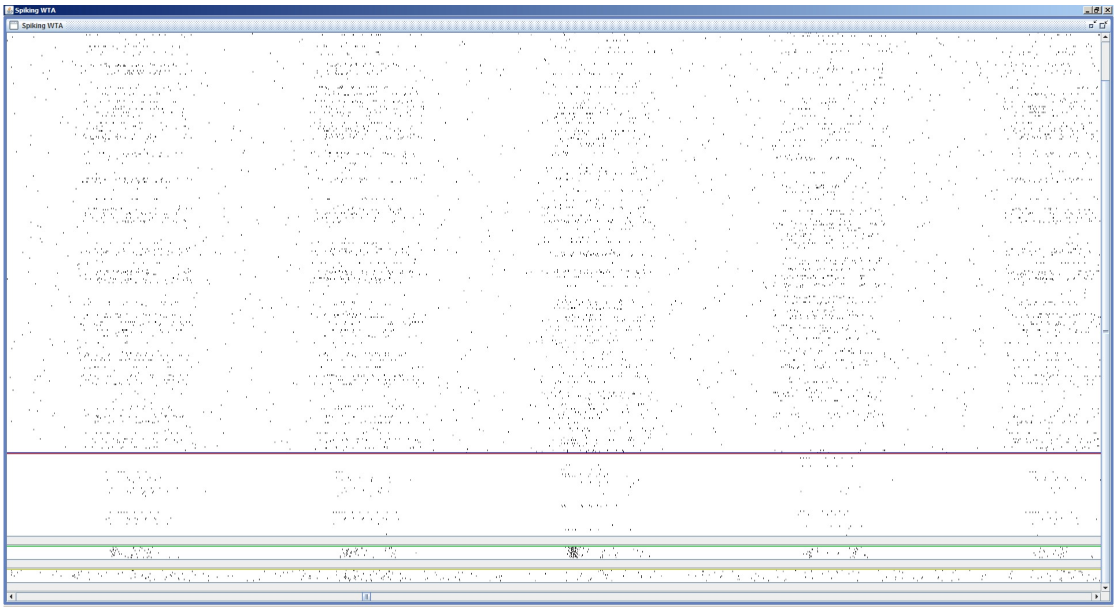

Figure A.2: The graphical interface illustrating the spike trains of all network units as realized by the `GUI` class.

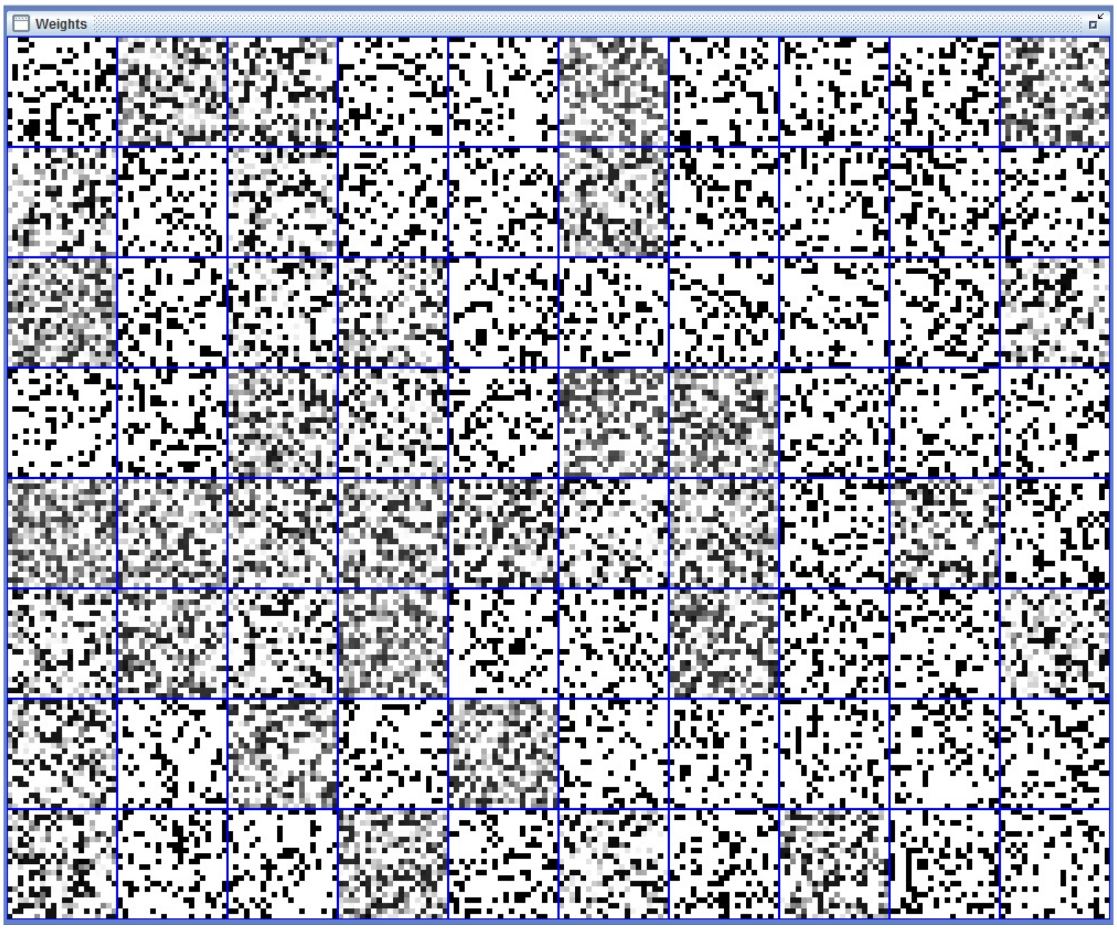

Figure A.3: The graphical interface illustrating weights of the network's z-units as realized by the `WeightMonitor` class.

A.3.5 Package `wta.evaluation`

The classes `ConditionalEntropyEvaluation` and `UpdateEvaluator` of this package provide the functionalities for evaluating the network's ouput after simulation as described in Chapter 4. `ConditionalEntropyEvaluation` realizes the sliding window evaluation over the simulation's duration and calculates the time course for the performance score (see Section 4.1.2) as well as for the conditional entropy (see Section 4.1.3).
`UpdateEvaluator` works similarly but uses `WeightUpdate` objects storing values and times of weight updates instead of the spike output of the network.
While `ConditionalEntropyEvaluation` evaluates the output right on instantiation `UpdateEvaluator` first requires the weight updates to be passed via the `storeUpdates()` method before starting the evaluation by calling the `eval()` method.

Appendix B

JavaDoc

Package wta

Class Summary

Network

Main class of the WTA simulation framework.

Population

Helper class to distribute neurons over CPU kernels.

---

wta

Class Network

```
java.lang.Object
   |
   +--java.awt.Component
         |
         +--java.awt.Container
               |
               +--java.awt.Window
                     |
                     +--java.awt.Frame
                           |
                           +--javax.swing.JFrame
                                 |
                                 +--wta.Network
```

All Implemented Interfaces:

java.awt.MenuContainer, java.awt.image.ImageObserver, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.TransferHandler.HasGetTransferHandler, javax.swing.WindowConstants

---

public class **Network**
extends javax.swing.JFrame

Main class of the WTA simulation framework. Abstracts a network of neurons in a WTA circuit. Provides functionality for configuration, simulation and evaluation of a network applied to a learning task.

Author:

Stefan Grabuschnig

Constructors

Network

```
public  Network(boolean batchMode)
```

Instantiates and initializes a network according to its configuration. If the number of trials specified in the main-method is larger than 1 the network is simulated in batch-mode without using graphical interfaces.

Parameters:

batchMode - determines if simulation is performed in batch-mode.

Methods

getJointEntropy

```
public double[] getJointEntropy()
```

Returns:

the joint entropy

---

getMeanCosSimilarityCurrent

```
public double[] getMeanCosSimilarityCurrent()
```

Returns:

cos similarities of weight updates to currently presented patterns

---

getMeanCosSimilarityFinal

```
public double[] getMeanCosSimilarityFinal()
```

Returns:

cos similarities of weight updates to the finally learned patterns

---

getNetEntropy

```
public double[] getNetEntropy()
```

Returns:

the normalized conditional entropy of the network output

---

getScore

```
public double[] getScore()
```

Returns:

the network score

---

main

```
public static void main(java.lang.String[] args)
```

---

saveOutput

```
public void saveOutput()
```

saves the output of the netowrk and evaluations.

---

sim

```
public void sim()
         throws java.lang.InterruptedException
```

Runs the simulation of the network.

Throws:

java.lang.InterruptedException -

---

writeOuptputFile

```
public static void writeOuptputFile(boolean[][] output,
                                    java.lang.String filename)
```

writes an output file for a matrix of boolean values.

Parameters:

output - the output
filename - the filename

---

writeOuptputFile

```
public static void writeOuptputFile(double[] output,
                                    java.lang.String filename)
```

writes an output file for an array of double values.

Parameters:

output - the output
filename - the filename

---

writeOuptputFileDouble

```
public static void writeOuptputFileDouble(double[][] output,
                                          java.lang.String filename)
```

writes an output file for a matrix of double values.

Parameters:

output - the output
filename - the filename

---

wta

Class Population

```
java.lang.Object
    |
    +--wta.Population
```

All Implemented Interfaces:

java.util.concurrent.Callable

---

public class **Population**
extends java.lang.Object
implements java.util.concurrent.Callable

Helper class to distribute neurons over CPU kernels.

Author:

Stefan Grabuschnig

Constructors

Population

```
public  Population(java.util.ArrayList population)
```

Defines a population of neurons.

Parameters:

population - the population of neurons

Methods

call

```
public java.lang.Boolean call()
```

Package wta.cell

Class Summary

BC

Represents a Basket Cell.

MC

Represents a Martinotti Cell.

Neuron

Represents an abstract neuron.

PC

Represents a pyramidal layer 5 neuron.

---

wta.cell

Class BC

```
java.lang.Object
  |
  +--wta.cell.Neuron
        |
        +--wta.cell.BC
```

---

public class **BC**
extends wta.cell.Neuron

Represents a Basket Cell. Provides the functionality for simulation of behavior and communication.

Author:

Stefan Grabuschnig

Constructors

BC

```
public  BC(int ID,
           int numPC)
```

Instantiates a Basket Cell.

Parameters:

ID - index of this Basket Cell
numPC - number of pyramidal cells in the network

Methods

getID

```
public int getID()
```

Returns:

the index of the BC

---

setA

```
public static void setA(double A)
```

sets the synaptic efficacy for synapses from PCs.

Parameters:

A - the synaptic efficacy

---

setDt

```
public static void setDt(double dt)
```

sets the size of a simulation time-step.

Parameters:

dt - size of a time-step in milliseconds

---

setVMShift

```
public static void setVMShift(double vMShift)
```

sets vMShift.

Parameters:

vMShift - the vMShift

---

update

```
public void update()
```

Overrides:

update in class wta.cell.Neuron

---

wta.cell

Class MC

```
java.lang.Object
    |
    +--wta.cell.Neuron
        |
        +--wta.cell.MC
```

---

public class **MC**
extends wta.cell.Neuron

Represents a Martinotti Cell. Provides the functionality for simulation of behavior and communication.

Author:

Stefan Grabuschnig

Constructors

MC

```
public  MC(int ID,
           int numPC)
```

Instantiates a Martinotti Cell.

Parameters:

ID - index of this Martinotti Cell
numPC - number of pyramidal cells in the network

Methods

getID

```
public int getID()
```

Returns:

the index of the BC

---

setA

```
public static void setA(double A)
```

sets the synaptic efficacy for synapses from PCs.

Parameters:

A - the synaptic efficacy

setDt

```
public static void setDt(double dt)
```

sets the size of a simulation time-step.

Parameters:

dt - size of a time-step in milliseconds

setVMShift

```
public static void setVMShift(double vMShift)
```

sets vMShift.

Parameters:

vMShift - the vMShift

update

```
public void update()
```

Overrides:

update in class wta.cell.Neuron

wta.cell

Class Neuron

```
java.lang.Object
   |
   +--wta.cell.Neuron
```

Direct Known Subclasses:

wta.cell.BC, wta.cell.MC, wta.cell.PC

public abstract class **Neuron**
extends java.lang.Object

Represents an abstract neuron. Defines an interface between the network simulator and individual neurons. Provides an interface for shared access to the spike output of neurons. Contains constants shared by all types of cells.

Author:

Stefan Grabuschnig

Constructors

Neuron

```
public  Neuron()
```

Methods

getOutputBC

```
public static boolean[][] getOutputBC()
```

Returns:

the output space for Basket Cells

---

getOutputMC

```
public static boolean[][] getOutputMC()
```

Returns:

the output space for Martinotti Cells

---

getOutputPC

```
public static boolean[][] getOutputPC()
```

Returns:

the output space of the z-units

---

getWeightUpdatesPC

```
public static double[][] getWeightUpdatesPC()
```

Returns:

the weight updates

---

nextTimeStep

```
public static void nextTimeStep()
```

sets the network state to the next time-step. Call after updating all neurons.

---

setDt

```
public static void setDt(double dt)
```

sets the size of simulation time-steps and updates all time-constants according to the change.

Parameters:

dt - size of a time-step

---

setOutputBC

```
public static void setOutputBC(boolean[][] outputBC)
```

sets the output space for Basket Cells.

Parameters:

outputBC - the output space for Basket Cells

---

setOutputMC

```
public static void setOutputMC(boolean[][] outputMC)
```

sets the output space for Martinotti Cells.

Parameters:

outputMC - the output space for Martinotti Cells

---

setOutputPC

```
public static void setOutputPC(boolean[][] outputPC)
```

sets the output space for z-units.

Parameters:

outputPC - the output space for z-units

---

setTimeStep

```
public static void setTimeStep(int timeStep)
```

sets all compartments of the network to a distinct simulation time-step.

Parameters:

timeStep - the time-step

---

setWeightUpdatesPC

```
public static void setWeightUpdatesPC(double[][] weightUpdatesPC)
```

sets the storage for weight updates.

Parameters:

weightUpdatesPC - storage for weight updates

---

update

```
public abstract void update()
```

updates all synapses, membrane potential and spike output of a neuron for a current time-step.

---

wta.cell

Class PC

```
java.lang.Object
     |
    +--wta.cell.Neuron
         |
        +--wta.cell.PC
```

---

public class **PC**
extends wta.cell.Neuron

Represents a pyramidal layer 5 neuron. Provides the functionality for simulation of behavior and communication.

Author:

Stefan Grabuschnig

Constructors

PC

```
public  PC(int ID,
           int numInputs,
           int numPC,
           int numBC,
           int numMC,
           double[] inputRates)
```

Instantiates a pyramidaly cell.

Parameters:

ID - unique index of the neuron
numInputs - number of input synapes
numPC - number of pyramidal cells in the network
numBC - number of Basket Cells in the network
numMC - number of Martinotti Cells in the network
inputRates - fraction of "high" neurons in input patterns

Methods

getID

```
public int getID()
```

Returns:

the index of the PC

---

getUpdateList

```
public java.util.ArrayList getUpdateList()
```

Returns:

the weight updates

---

getWeights

```
public double[] getWeights()
```

Returns:

the synaptic weights

---

setA_BCtoPC

```
public static void setA_BCtoPC(double A_BCtoPC)
```

sets the synaptic efficacy for synapses from Basket Cells.

Parameters:

A_BCtoPC - the synaptic efficacy

---

setA_MCtoPC

```
public static void setA_MCtoPC(double A_MCtoPC)
```

sets the synaptic efficacy for synapses from Martinotti Cells.

Parameters:

A_MCtoPC - the synaptic efficacy

---

setBl

```
public static void setBl(double bl)
```

sets the homeostasis target rate.

Parameters:

bl - the homeostasis target rate

---

setDt

```
public static void setDt(double dt)
```

sets the size of a simulation time-step.

Parameters:

dt - size of time-step in milliseconds

---

setEta

```
public static void setEta(double eta)
```

sets the learning rate.

Parameters:

eta - the learning rate

---

setNextBatch

```
public static void setNextBatch(boolean[][] batch)
```

sets the next batch of spike input.

Parameters:

batch - batch of spiek input

---

update

```
public void update()
```

Overrides:

update in class wta.cell.Neuron

Package wta.evaluation

Class Summary

ConditionalEntropyEvaluation

This class provides functionality for evaluating the network's performance and convergence.

UpdateEvaluator

This class provides functionality for evaluating the cos-similarities of the network's weight updates.

WeightUpdate

Stores values and time of the weight update of a z-unit.

---

wta.evaluation

Class ConditionalEntropyEvaluation

```
java.lang.Object
   |
   +--wta.evaluation.ConditionalEntropyEvaluation
```

---

public class **ConditionalEntropyEvaluation**
extends java.lang.Object

This class provides functionality for evaluating the network's performance and convergence.

Author:

Stefan Grabuschnig

Constructors

ConditionalEntropyEvaluation

```
public  ConditionalEntropyEvaluation(int numPatterns,
                                     int patternLength,
                                     boolean[][] outputPC,
                                     int[] patternSequence)
```

Instantiates the evluator.

Parameters:

numPatterns - number of patterns
patternLength - length of pattern presentation in time-steps
outputPC - spike output of the pyramidal cells
patternSequence - sequence of pattern presentations

Methods

getBest

```
public double[][] getBest()
```

Returns:

the time courses of the specificities of the best winner neurons of each pattern

---

getBestEntropies

```
public double[][] getBestEntropies()
```

Returns:

time course of the conditional entropies for the winner neurons

---

getJointEntropy

```
public double[] getJointEntropy()
```

Returns:

time course of the joint entropy of the network

---

getNetEntropy

```
public double[] getNetEntropy()
```

Returns:

time course of the conditional entropy of the network

---

getScore

```
public double[][] getScore()
```

Returns:

scores of the winner neurons

---

getWinners

```
public int[] getWinners()
```

Returns:

IDs of the winner neurons

---

wta.evaluation

Class UpdateEvaluator

```
java.lang.Object
    |
    +--wta.evaluation.UpdateEvaluator
```

---

public class **UpdateEvaluator**
extends java.lang.Object

This class provides functionality for evaluating the cos-similarities of the network's weight updates.

Author:

Stefan Grabuschnig

Constructors

UpdateEvaluator

```
public  UpdateEvaluator(int numCells,
                        int[] patternSequence,
                        int patternLength,
                        int outputLength,
                        java.util.ArrayList patterns)
```

Instantiates a weight update evaluator.

Parameters:

numCells - number of pyramidal cells
patternSequence - sequence of pattern presentations
patternLength - duration of a pattern presentation
outputLength - length of the output in time-steps
patterns - the patterns of the learning task

Methods

eval

`public void eval(int[] best)`

starts the evaluation of the weight updates of the best winner neurons for each pattern.

Parameters:

best - the best neurons for each pattern

---

getCosSimilaritiesCurrent

`public double[][] getCosSimilaritiesCurrent()`

Returns:

cos similarities of weight updates to currently presented patterns

---

getCosSimililaritiesFinal

`public double[][] getCosSimililaritiesFinal()`

Returns:

cos similarities of weight updates to the finally learned patterns

---

getMeanCosSimilaritiesCurrent

`public double[] getMeanCosSimilaritiesCurrent()`

Returns:

average cos similarities of weight updates to currently presented patterns over the winners

---

getMeanCosSimilaritiesFinal

`public double[] getMeanCosSimilaritiesFinal()`

Returns:

average cos similarities of weight updates to the finally learned patterns over the winners

---

getUpdateTimes

```
public double[][] getUpdateTimes()
```

Returns:

the times of the weight updates

---

storeUpdates

```
public void storeUpdates(int cell,
                         java.util.ArrayList updateList)
```

uptake of weight updates from individual cells.

Parameters:

cell - ID of the respective cell
updateList - weight updates of the respective cell

---

wta.evaluation

Class WeightUpdate

```
java.lang.Object
    |
    +--wta.evaluation.WeightUpdate
```

---

public class **WeightUpdate**
extends java.lang.Object

Stores values and time of the weight update of a z-unit.

Author:

Stefan Grabuschnig

Constructors

WeightUpdate

```
public  WeightUpdate(int t,
                     double[] update)
```

Instantiates a weight update.

Parameters:

t - time of the update
update - values of the update

Methods

getT

```
public int getT()
```

Returns:

the time of the update

---

getUpdate

```
public double[] getUpdate()
```

Returns:

the weight update

Package wta.gui

Class Summary

GUI

The graphical interface, displaying all input and output spikes of the network on different panels, where each type of unit has its own panel.

WeightMonitor

A window displaying the current state of each of the network's z-units weights represented by grayscale pixels arranged in a blue frame, where black indicates a high weight and white a weight close to zero.

---

wta.gui

Class GUI

```
java.lang.Object
    |
    +--java.awt.Component
        |
        +--java.awt.Container
            |
            +--javax.swing.JComponent
                |
                +--javax.swing.JInternalFrame
                    |
                    +--wta.gui.GUI
```

All Implemented Interfaces:

java.awt.MenuContainer, java.awt.image.ImageObserver, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.TransferHandler.HasGetTransferHandler, javax.swing.WindowConstants

---

public class **GUI**
extends javax.swing.JInternalFrame

The graphical interface, displaying all input and output spikes of the network on different panels, where each type of unit has its own panel. The implementation currently includes panels for input units, z-units representing pyramidal cells, units representing Basket Cells and units representing Martinotti Cells.

Author:

Stefan Grabuschnig

Constructors

GUI

`public  GUI(wta.input.Input input)`

Creates and initializes the graphical interface, depending on the size of the network input.

Parameters:

input - Instance of the spike input of a network

Methods

drawBCOutput

`public void drawBCOutput(boolean[][] batch)`

Draws a batch of output spikes on the panel representing the Basket Cell's output space.

Parameters:

batch - a batch of binary spike output

---

drawInput

`public void drawInput()`

Draws the network's spike input on the respective panel.

---

drawMCOutput

`public void drawMCOutput(boolean[][] batch)`

Draws a batch of output spikes on the panel representing the Martinotti Cell's output space.

Parameters:

batch - a batch of binary spike output

---

drawZOutput

`public void drawZOutput(boolean[][] batch)`

Draws a batch of output spikes on the panel representing the z-unit's output space.

Parameters:

batch - a batch of binary spike output

---

wta.gui

Class WeightMonitor

```
java.lang.Object
    |
    +--java.awt.Component
        |
        +--java.awt.Container
            |
            +--javax.swing.JComponent
                |
                +--javax.swing.JInternalFrame
                    |
                    +--wta.gui.WeightMonitor
```

All Implemented Interfaces:

java.awt.MenuContainer, java.awt.image.ImageObserver, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.TransferHandler.HasGetTransferHandler, javax.swing.WindowConstants

---

public class **WeightMonitor**
extends javax.swing.JInternalFrame

A window displaying the current state of each of the network's z-units weights represented by grayscale pixels arranged in a blue frame, where black indicates a high weight and white a weight close to zero. The update method has to be called in order to update the content of the window.

Author:

Stefan Grabuschnig

Constructors

WeightMonitor

`public  WeightMonitor(java.util.ArrayList zLayer)`

Instantiates a WeightMonitor window for an ArrayList of z-units.

Parameters:

zLayer - ArrayList containing the z-units, whose weights shall be monitored.

Methods

update

`public void update()`

Draws the current state of the weights.

Package wta.input

Class Summary

Input

Represents the spike input of a network.

InputBars

Type of network input, where the input units are rectangular arranged and patterns are comprised by bars of "high" input units, a horizontal and a vertical one each.

InputRandomBars

Type of network input, where the input units are rectangular arranged and patterns are comprised by a bars of "high" input units.

InputRandomBarsNoOverlap

Type of network input, where the input units are rectangular arranged and patterns are comprised by a bars of "high" input units.

---

wta.input

Class Input

```
java.lang.Object
   |
   +--wta.input.Input
```

Direct Known Subclasses:

wta.input.InputBars, wta.input.InputRandomBars

---

public class **Input**
extends java.lang.Object

Represents the spike input of a network. Provides functionality for generation and managment of the network's input. Randomly creates different patterns of "high" and "low" input units. Randomly creates a sequence of pattern presentations with different classes of patterns with different probabilities to be presented to network.

Author:

Stefan Grabuschnig

Constructors

Input

```
public  Input(int simDuration,
              int dt,
              int numInputs,
              int numPatterns,
              int inputDuration,
              double[] patternProbs,
              java.lang.Double inputRateHigh,
              java.lang.Double inputRateLow,
              java.lang.Double inputRateLow2,
              double[] inputRates,
              double rateScale)
```

Instantiates and sets up the spike input of a network.

Parameters:

simDuration - number of time-steps of the simulation
dt - size of a time-step in milliseconds
numInputs - number of input units
numPatterns - number of patterns to be generated
inputDuration - number of time-steps a pattern is presented to the network
patternProbs - Array containing probabilities for each class of pattern
inputRateHigh - spiking frequency for "high" input units
inputRateLow - spiking frequency for "low" input units
inputRateLow2 - parameter for the shape of a "high" unit's spike train
inputRates - fraction of "high" neurons
rateScale - scaling factor for spiking rates

Methods

getBatch

```
public boolean[][] getBatch()
```

Returns:

sequentially returns a batch of spike input each time the method is called

---

getInput

```
public java.util.ArrayList getInput()
```

Returns:

the input spike trains

---

getInputDuration

```
public int getInputDuration()
```

Returns:

the inputDuration

---

getNumInputs

```
public int getNumInputs()
```

Returns:

the numInputs

---

getPatternSequence

```
public int[] getPatternSequence()
```

Returns:

the sequence of pattern presentations

---

getPatterns

```
public java.util.ArrayList getPatterns()
```

Returns:

the patterns

---

getSimDuration

```
public int getSimDuration()
```

Returns:

the simDuration

---

saveInput

```
public void saveInput()
```

writes the input spike trains to a text-file "input.txt".

wta.input

Class InputBars

```
java.lang.Object
    |
    +--wta.input.Input
        |
        +--wta.input.InputBars
```

public class **InputBars**
extends wta.input.Input

Type of network input, where the input units are rectangular arranged and patterns are comprised by bars of "high" input units, a horizontal and a vertical one each.

Author:

Stefan Grabuschnig

Constructors

InputBars

```
public  InputBars(int simDuration,
                  int dt,
                  int numInputs,
                  int numPatterns,
                  int inputDuration,
                  double[] patternProbs,
                  java.lang.Double inputRateHigh,
                  java.lang.Double inputRateLow,
                  java.lang.Double inputRateLow2,
                  double[] inputRates,
                  double rateScale)
```

Instantiates and sets up the spike input of a network.

Parameters:

simDuration - number of time-steps of the simulation
dt - size of a time-step in milliseconds
numInputs - number of input units
numPatterns - number of patterns to be generated
inputDuration - number of time-steps a pattern is presented to the network
patternProbs - Array containing probabilities for each class of pattern
inputRateHigh - spiking frequency for "high" input units
inputRateLow - spiking frequency for "low" input units
inputRateLow2 - parameter for the shape of a "high" unit's spike train
inputRates - fraction of "high" neurons
rateScale - scaling factor for spiking rates

wta.input

Class InputRandomBars

```
java.lang.Object
    |
    +--wta.input.Input
        |
        +--wta.input.InputRandomBars
```

Direct Known Subclasses:

wta.input.InputRandomBarsNoOverlap

---

public class **InputRandomBars**
extends wta.input.Input

Type of network input, where the input units are rectangular arranged and patterns are comprised by a bars of "high" input units. A horizontal and a vertical bar are randomly generated each time a pattern is presentation to the network.

Author:

Stefan Grabuschnig

Constructors

InputRandomBars

```
public  InputRandomBars(int simDuration,
                        int dt,
                        int numInputs,
                        int numPatterns,
                        int inputDuration,
                        double[] patternProbs,
                        java.lang.Double inputRateHigh,
                        java.lang.Double inputRateLow,
                        java.lang.Double inputRateLow2,
                        double[] inputRates,
                        double rateScale)
```

Instantiates and sets up the spike input of a network.

Parameters:

simDuration - number of time-steps of the simulation
dt - size of a time-step in milliseconds
numInputs - number of input units
numPatterns - number of patterns to be generated
inputDuration - number of time-steps a pattern is presented to the network
patternProbs - Array containing probabilities for each class of pattern
inputRateHigh - spiking frequency for "high" input units
inputRateLow - spiking frequency for "low" input units
inputRateLow2 - parameter for the shape of a "high" unit's spike train
inputRates - fraction of "high" neurons
rateScale - scaling factor for spiking rates

wta.input

Class InputRandomBarsNoOverlap

```
java.lang.Object
   |
   +--wta.input.Input
         |
         +--wta.input.InputRandomBars
               |
               +--wta.input.InputRandomBarsNoOverlap
```

public class **InputRandomBarsNoOverlap**
extends wta.input.InputRandomBars

Type of network input, where the input units are rectangular arranged and patterns are comprised by a bars of "high" input units. A horizontal and a vertical bar are randomly generated each time a pattern is presentation to the network, with the restriction that there exists no overlapping of bars of the same orientation.

Author:

Stefan Grabuschnig

Constructors

InputRandomBarsNoOverlap

```
public  InputRandomBarsNoOverlap(int simDuration,
                                 int dt,
                                 int numInputs,
                                 int numPatterns,
                                 int inputDuration,
                                 double[] patternProbs,
                                 java.lang.Double inputRateHigh,
                                 java.lang.Double inputRateLow,
                                 java.lang.Double inputRateLow2,
                                 double[] inputRates,
                                 double rateScale)
```

Instantiates and sets up the spike input of a network.

Parameters:

simDuration - number of time-steps of the simulation
dt - size of a time-step in milliseconds
numInputs - number of input units
numPatterns - number of patterns to be generated
inputDuration - number of time-steps a pattern is presented to the network
patternProbs - Array containing probabilities for each class of pattern
inputRateHigh - spiking frequency for "high" input units
inputRateLow - spiking frequency for "low" input units
inputRateLow2 - parameter for the shape of a "high" unit's spike train
inputRates - fraction of "high" neurons
rateScale - scaling factor for spiking rates

Bibliography

[1] Gerstner, Wulfram and Kistler, Werner. *Spiking Neuron Models.* Cambridge University Press, New York, NY, USA, 2002. ISBN 0521890799.

[2] Arne Vladimir Blackman, Stefan Grabuschnig, Robert Legenstein and Per Jesper Sjöström. A Comparison of Manual Neuronal Reconstruction from Biocytin Histology or 2-Photon Imaging: Morphometry and Computer Modeling. *Frontiers in Neuroanatomy*, 8(65), 2014. ISSN 1662-5129. doi: 10.3389/fnana.2014.00065.

[3] Amarasingham, Asohan and Chen, Ting-Li and Geman, Stuart and Harrison, Matthew T and Sheinberg, David L. Spike count reliability and the Poisson hypothesis. *The Journal of neuroscience*, 26(3):801–809, 2006.

[4] Perkel, Donald H and Gerstein, George L and Moore, George P. Neuronal spike trains and stochastic point processes: I. The single spike train. *Biophysical journal*, 7(4):391–418, 1967.

[5] Averbeck, Bruno B. Poisson or not poisson: differences in spike train statistics between parietal cortical areas. *Neuron*, 62(3):310–311, 2009.

[6] Heeger, David. Poisson model of spike generation. *Handout, University of Standford*, 5, 2000.

[7] Nelson, Sacha B. Cortical Microcircuits-Diverse or Canonical? *Neuron*, 36(1): 19–27, 2002.

[8] Douglas, Rodney J and Martin, Kevan AC. Neuronal circuits of the neocortex. *Annu. Rev. Neurosci.*, 27:419–451, 2004.

[9] Mountcastle, Vernon B. The columnar organization of the neocortex. *Brain*, 120 (4):701–722, 1997.

[10] Markram, H. and ToledoRodriguez, M. and Wang, Y. and Gupta, A. and Silberberg, G. and Wu, C. Interneurons of the neocortical inhibitory system. *Nature Reviews Neuroscience*, 5(10):793–807, 2004.

[11] Maass, Wolfgang. On the computational power of winner-take-all. *Neural computation*, 12(11):2519–2535, 2000.

[12] Rosenblatt, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[13] Bishop, Christopher M. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[14] Nessler, Bernhard and Pfeiffer, Michael and Buesing, Lars and Maass, Wolfgang. Bayesian Computation Emerges in Generic Cortical Microcircuits through Spike-Timing-Dependent Plasticity. *PLoS Comput Biol*, 9(4):e1003037+, April 2013. doi: 10.1371/journal.pcbi.1003037.

[15] Habenschuss, Stefan and Puhr, Helmut and Maass, Wolfgang. Emergence of optimal decoding of population codes through STDP. *Neural computation*, 25(6):1371–1407, 2013.

[16] Stefan Habenschuss and Johannes Bill and Bernhard Nessler. Homeostatic plasticity in Bayesian spiking networks as Expectation Maximization with posterior constraints. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 773–781. Curran Associates, Inc., 2012.

[17] Nessler, Bernhard and Pfeiffer, Michael and Maass, Wolfgang. STDP enables spiking neurons to detect hidden causes of their inputs. In *Advances in neural information processing systems*, pages 1357–1365, 2009.

[18] LeCun, Yann and Bottou, Léon and Bengio, Yoshua and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

[19] Palmer, Lucy and Murayama, Masanori and Larkum, Matthew. Inhibitory Regulation of Dendritic Activity in vivo. *Front Neural Circuits*, 6:26, 2012. ISSN 1662-5110.

[20] Palmer, Lucy M. and Schulz, Jan M. and Murphy, Sean C. and Ledergerber, Debora and Murayama, Masanori and Larkum, Matthew E. The cellular basis of GABA(B)-mediated interhemispheric inhibition. *Science (New York, N.Y.)*, 335(6071):989–93, February 2012. doi: 10.1126/science.1217276.

[21] Blackman, Arne V and Abrahamsson, Therese and Costa, Rui P. and Lalanne, Txomin and Sjöström, Per Jesper. Target Cell-Specific Short-Term Plasticity in

Local Circuits. *Frontiers in Synaptic Neuroscience*, 5(11), 2013. ISSN 1663-3563. doi: 10.3389/fnsyn.2013.00011.

[22] Misha Tsodyks and Klaus Pawelzik and Henry Markram and M. Tsodyks. Neural Networks with Dynamic Synapses. *Neural Computation*, 10:821–835, 1998.

[23] Tsodyks, Misha V and Markram, Henry. The neural code between neocortical pyramidal neurons depends on neurotransmitter release probability. *Proceedings of the National Academy of Sciences*, 94(2):719–723, 1997.

[24] Tsodyks, Misha V and Markram, Henry. Plasticity of neocortical synapses enables transitions between rate and temporal coding. In *Artificial Neural Networks—ICANN 96*, pages 445–450. Springer, 1996.

[25] Costa, Rui P and Sjöström, P Jesper and van Rossum, Mark C W. Probabilistic inference of short-term synaptic plasticity in neocortical microcircuits. *Front Comput Neurosci*, 7:75, 2013. ISSN 1662-5188.

[26] Spruston, Nelson. Pyramidal neurons: dendritic structure and synaptic integration. *Nature Reviews Neuroscience*, 9(3):206–221, 2008.

[27] Polsky, Alon and Mel, Bartlett W and Schiller, Jackie. Computational subunits in thin dendrites of pyramidal cells. *Nature neuroscience*, 7(6):621–627, 2004.

[28] Kampa, B. and Letzkus, J. and Stuart, G. Dendritic mechanisms controlling spike-timing-dependent synaptic plasticity. *Trends in Neurosciences*, 30(9):456–463, September 2007. ISSN 01662236. doi: 10.1016/j.tins.2007.06.010.

[29] Spruston, Nelson and Kath, William L. Dendritic arithmetic. *Nature neuroscience*, 7(6):567–569, 2004.

[30] Matthew Larkum. A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends in Neurosciences*, 36(3):141–151, March 2013. ISSN 01662236. doi: 10.1016/j.tins.2012.11.006.

[31] Stefan Häusler and Wolfgang Maass. Inhibitory networks orchestrate the self-organization of computational function in cortical microcircuit motifs through STDP. in preparation, November 2013.

[32] Larkum, Matthew E. and Nevian, Thomas and Sandler, Maya and Polsky, Alon and Schiller, Jackie. Synaptic Integration in Tuft Dendrites of Layer 5 Pyramidal Neurons: A New Unifying Principle. *Science*, 325(5941):756–760, August 2009. ISSN 1095-9203. doi: 10.1126/science.1171958.

[33] Larkum, Matthew E. and Zhu, Julius J. and Sakmann, Bert. A new cellular mechanism for coupling inputs arriving at different cortical layers. *Nature*, 398(6725): 338–341, March 1999. doi: 10.1038/18686.

[34] Larkum, M. E. and Kaiser, K. M. and Sakmann, B. Calcium electrogenesis in distal apical dendrites of layer 5 pyramidal cells at a critical frequency of back-propagating action potentials. *Proceedings of the National Academy of Sciences of the United States of America*, 96(25):14600–14604, December 1999. ISSN 0027-8424.

[35] Ledergerber, Debora and Larkum, Matthew Evan. The time window for generation of dendritic spikes by coincidence of action potentials and EPSPs is layer specific in somatosensory cortex. *PLoS One*, 7(3):e33146, 2012. ISSN 1932-6203.

[36] Kampa, Björn M. and Letzkus, Johannes J. and Stuart, Greg J. Requirement of dendritic calcium spikes for induction of spike-timing-dependent synaptic plasticity. *The Journal of physiology*, 574(Pt 1):283–290, July 2006. ISSN 0022-3751. doi: 10.1113/jphysiol.2006.111062.

[37] Hines, M. L. and Carnevale, N. T. The NEURON Simulation Environment. *Neural Computation*, 9(6):1179–1209, August 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.6.1179.

[38] Stuart, G. J. and Häusser, M. Dendritic coincidence detection of EPSPs and action potentials. *Nat Neurosci*, 4(1):63–71, January 2001. ISSN 1097-6256. doi: 10.1038/82910.

[39] Izhikevich, E. M. and Desai, N. S. and Walcott, E. C. Bursts as a unit of neural information: selective communication via resonance. *Trends Neurosci*, 26:161–167, 2003.

[40] P. Földiák and P. Fiildihk. Forming sparse representations by local anti-Hebbian learning, 1990.

[41] Triesch, Jochen. Synergies Between Intrinsic and Synaptic Plasticity Mechanisms. *Neural Computation*, 19(4):885–909, 2007.

[42] Zeno Jonke. *Stochastic Computations and Learning in Networks of Spiking Neurons: Simulation framework, Analysis and Theory.* PhD thesis, Graz University of Technology, Institute for Theoretical Computer Sciences, 2013.

[43] Krieg, Daniel and Triesch, Jochen. A unifying theory of synaptic long-term plasticity based on a sparse distribution of synaptic strength. *Frontiers in Synaptic Neuroscience*, 6(3), 2014. ISSN 1663-3563. doi: 10.3389/fnsyn.2014.00003.

[44] Naud, Richard and Bathellier, Brice and Gerstner, Wulfram. Spike-timing prediction in cortical neurons with active dendrites. *Frontiers in computational neuroscience*, 8, 2014.

[45] Ho, Victoria M and Lee, Ji-Ann and Martin, Kelsey C. The cell biology of synaptic plasticity. *Science*, 334(6056):623–628, 2011.