On Competition and Learning in Cortical Structures

by Florian Jug

Diss. ETH No. 20194

On Competition and Learning in Cortical Structures

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
Florian Jug
Dipl. inf. TU
born 08.04.1979
citizen of Austria

accepted on the recommendation of
Prof. Dr. Angelika Steger, examiner
Dr. Matthew Cook, co-examiner
Dr. Piotr Dudek, co-examiner
Prof. Dr. Kevan Martin, co-examiner

January 13th 2012

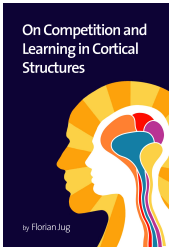
The official ISBN numbers for this thesis are:

Hardcover
ISBN 978-3-906031-02-6

Softcover
ISBN 978-3-906031-03-3

eBook
ISBN 978-3-906031-04-0

Cover art inspired by a CD cover of the wonderful trip-hop duo Zero7 and skillfully created by Gaia Pigino.



The following photo was shot by Rasto Sramek. It shows me, only minutes after my defense on Friday, January 13th 2012 at ETH Zurich.

To family and friends.

Contents

Abstract

How do our brains make us live our lives? Despite the tremendous progress life sciences have made in the last decades we have no conclusive answer to this question. Today we are, however, able to build machines that outperform us in many ways. We can't calculate as fast as our machines do, nor can we win against them when playing Backgammon, Chess, or Jeopardy. The paradigms upon which they function are fundamentally different from what we know about neural computation. Not only the computational substrate, but the very nature of human intelligence seems fundamentally different from the artificial intelligence (AI) we put into our machines.

Existing computational models can be subdivided in two broad classes: (*i*) top-down models like the AI examples mentioned above that aim to reproduce high-level abilities, without worrying whether the model is in any way similar to the implementation in our brains, and (*ii*) bottom-up models that focus on using biologically plausible modules that are based on established neurophysiological and neuroanatomical knowledge. The ultimate goal is to understand the detailed neural correlates of cognition – a goal that is currently out of reach.

In the first part of this thesis we use classical models for cortical competition to build adaptive systems that are capable of learning the relation between two input modalities. We go on to show that a similar system can be used to sharpen the connectivity in a network of interconnected areas.

In the second part of this thesis we replace the classical competition models used so far by a class of recurrent networks that we call Recurrent Competitive Networks (RCNs). After a thorough literature review to deduce biologically plausible parameter regimes for spiking network simulations we study the dynamics of spiking RCNs and show that some features of classical competition models (CCMs) happen to arise naturally in RCNs.

For the sake of faster simulations we introduce and switch to a suitable higher level of abstraction on the basis of rate coded Siegert neurons. After we show that the dynamics of spiking RCNs are well conserved when using Siegert neurons, we reintroduce plasticity mechanisms like the ones used in the first two chapters.

In the third part of this thesis we show that adaptive, plastic RCNs are capable of learning the topology of their inputs, a feature that distinguishes these networks from CCMs, which are by design predetermined for one fixed topology. In the last chapter we finally describe a system of coupled RCNs that is capable of learning and using (*i*) the relation between previously unknown inputs, presented to different parts of the network, and (*ii*) the input encoding (input topology) simultaneously.

In summary we present novel bottom-up models that are capable of adapting to a-priori unknown environments with unknown input encodings. The proposed networks do not have to be tuned in order to make sense of their inputs – they just do what they are supposed to do. The type of networks we propose are likely to exist in many places in the brains of many species, but we tend to see them as models of localized structures in mammalian cortices.

Although we are not yet at the point where we have a scalable, cognitive model that can learn to solve complex real-world problems, we believe this thesis makes concrete contributions that bring us one step closer to this goal.

Zusammenfassung

Auf welche Weise hilft uns unser Gehirn erfolgreich durchs Leben zu gehen? Trotz des unglaublichen Fortschritts der Biowissenschaften in den letzten Jahrzehnten können wir diese Frage nicht zufriedenstellend beantworten. Dennoch sind wir heute in der Lage Maschinen zu bauen, die uns selbst in vielen Dingen überlegen sind. Diese können diese schneller rechnen als wir, sie schlagen uns auch beim Backgammon, Schach, und sogar beim Jeopardy. Da wir wissen, wie diese Maschinen arbeiten, ist uns aber auch klar, dass die Paradigmen, auf denen ihr Erfolg beruht, fundamentale Unterschiede zur Funktionsweise unseres Gehirns aufweisen. Nicht nur die eingesetzte Hardware, auch die Natur menschlicher Intelligenz an und für sich scheint sich grundsätzlich von der künstlichen Intelligenz (KI), die wir unseren Maschinen beizubringen vermögen, zu unterscheiden.

Existierende Herangehensweisen können in zwei Klassen unterteilt werden: (*i*) top-down Modelle, wie die eben genannten KI Ansätze, mit denen versucht wird, eine bestimmte Fertigkeit zu erlangen, ohne darauf Rücksicht nehmen zu müssen, ob die dem System zugrundeliegende Funktionsweise jener unseres Gehirns entspricht sowie (*ii*) bottom-up Modelle, deren Ziel es ist, ausschliesslich biologisch plausible Bausteine miteinander zu kombinieren. Das ultimative Ziel dabei ist, ein detailliertes Verständnis über das neuronale Korrelat menschlicher Kognition zu erlangen – ein Ziel, dem wir nicht sehr nahe zu sein scheinen.

Im ersten Teil dieser These benutzen wir klassische Modelle kortikaler Konkurrenz (KKM), die den Wettbewerb von Nervenzellen, im neuronalen Verbund aktiv zu sein, modellieren. Wir benutzen diese, um adaptive Systeme zu bauen, die Relationen zwischen zwei Inputsignalen zu lernen vermögen. In einem zweiten Ergebnis zeigen wir, dass ein ähnliches System die biologisch gewachsenen Verbindungen zwischen Gehirnarealen präzisieren kann.

Im zweiten Teil dieser Arbeit ersetzen wir die bisher verwendeten KKM mit Instanzen einer bestimmten Klasse rekurrenter Netze (RN). Nach einer detaillierten Besprechung der verfügbaren Literatur leiten wir biologisch plausible Wertebereiche für die in unseren Simulationen benötigten Parameter ab. Diese benutzen wir dann, um die Dynamiken von feuernden RN zu studieren. Wir zeigen, dass gewisse Dynamiken von RN denen der klassischen Konkurrenzmodelle ähneln. 1

Um unsere Simulationen zu beschleunigen, führen wir ein höher abstrahiertes Neuronenmodell ein: das Siegert neuron. Nachdem wir zeigen, dass die Dynamiken feuernder RN mit denen von Siegert-RN sehr gut übereinstimmen, benutzen wir dieses abstrahierte Modell und erweitern es um Lern- und Plastizitätsmechanismen, die wir in sehr ähnlicher Form bereits in den ersten Kapiteln angewandt haben.

Wir zeigen, dass adaptive, plastische RN in der Lage sind, die Topologie, der ihnen zugeführten Inputs, zu erlernen, ein Feature, welches diese Netzwerke auszeichnet und stark von KKM abhebt. Letztere obliegen per Definition einer fixen, zum Erzeugungszeitpunkt festzulegenden Topologie.

Im letzten Kapitel beschreiben wir ein System, welches mehrere plastische RN miteinander kombiniert und in der Lage ist, zur gleichen Zeit (*i*) die Relation zwischen zuvor unbekannten Inputs, sowie (*ii*) die Kodierung dieser Inputs (deren Topologie) zu verstehen und nutzen zu lernen.

Zusammenfassend präsentieren wir neuartige bottom-up Modelle, die in der Lage sind, sich an a-priori unbekannte Umgebungen anzupassen. Die vorgeschlagenen Netzwerke müssen nicht speziell für bestimmte Umgebungen gebaut werden, sondern sie passen sich selbständig an diese an. Die Klasse von Netzwerken, die wir dazu benutzen, kann in Gehirnen vieler Arten gefunden werden. Wir tendieren jedoch dazu, unsere Netze als Modelle für lokale, kortikale Strukturen in Gehirnen von Säugetieren zu sehen.

Obwohl wir noch nicht an dem Punkt angelangt sind, ein skalierbares, kognitives Modell zu haben, welches von sich aus in der Lage ist, komplexe realweltliche Probleme zu lösen, sind wir uns sicher, mit dieser Arbeit einen wertvollen Schritt in diese Richtung zu machen.

Acknowledgments

First of all I want to thank my supervisor Angelika Steger. She gave me a probably unique possibility to pursue a novel type of project in her group. This, in fact, gave me the chance too experience the bitter sweet combination of having the freedom to choose topics and the responsibility to choose them wisely. I very much enjoyed both our scientific collaboration and our conversations about academic topics in a broader sense.

An especially big "thank you" goes to Matthew Cook. Not only my thesis would be very different, but my entire scientific development would most likely have gone in a very different direction without him. I feel lucky to have met him as early as I did.

I am grateful to Kevan Martin for agreeing to act as the co-examiner of this thesis, and for countless tasty breakfasts that nurtured not only my body but mainly my mind. I learned many very important lessions about neuroanatomy, neurophysiology, experimental techniques, science, and scientists in his precious group meetings.

Next on my list I want to thank the last of my co-examiners, Piotr Dudek. His careful reading lead to important feedback for this final version of my thesis. I hope that the future will bring the possibility to collaborate with him and his group.

For more than two semesters I enjoyed several hours a week in the group meetings of Rodney Douglas. It was an important period which taught me a lot about neuroscience, self organization, neural development, and

neuromorphic engineering. I am very grateful for having had the change to be part of his team.

A very special thank you goes to all past and present members of the ever-changing CSA family. Despite being a (scientific) outlier I felt at home in our group from the beginning, and many of you have become dear friends over the years. So, thank you Nicla Bernasconi, Joshua Cooper, Stefanie Gerke, Luca Gugelmann, Dan Hefetz, Christoph Krautz, Fabian Kuhn, Julian Lorenz, Johannes Lengler, Martin Marciniszyn, Torsten Mütze, Konstantinos Panagiotou, Ueli Peter, Thomas Rast, Jan Remy, Justus Schwartz, Alexander Souza, Reto Spöhel, Henning Thomas, and Andreas Weißl. I also want to thank Yves Brise, Tobi Christ, Yann Disser, Matus Mihalak, Robin Moser, Dominik Scheder, Marcel Schöngens, Rasto Sramek, Marek Sulovsky, and Anka Zych from the other two CADMO groups for many enjoyable hours spent together. Two of all the mentioned people I need to thank explicitly: Christoph, who was by far the closest collaborator and friend during our PhD time, and Johannes, who spent hours reading and commenting on early versions of this thesis.

But as mentioned before, not only the people sitting close to me, but also the people at my second scientific home, the INI, deserve special thank for all the hours of fruitful interaction. Thank you Roman Bauer, Nuno da Costa, Tobi Delbruck, Jan Funke, Stephan Gerhard, Dennis Göhlsdorf, Andreas Hauri, Giacomo Indiveri, Andi Keller, Dylan Muir, Emre Neftci, Marco Perella, Michael Pfeiffer, Sabina Pfister, Jason Rolfe, Sylvia Schöder, Isabelle Spühler, and Andreas Steimer.

Further I want to thank my family for their love and support, and my friends in Munich, Vienna, and Zurich for reasons that are to numerous to list here. I am glad you are there.

Last but certainly not least I have to thank my wife Gaia. Without her I might not have survived the last month of my theses (which coincided with the first month of our time as married couple). At times when even I have a hard time tolerating myself, she gives me the power to move forwards.

CHAPTER 1

Introduction and Overview[1]

Artificial Intelligence

How do we think, reason, love, or write a thesis? How do our brains make us live our lives?

Many of us are interested in these types of questions, but nobody has yet found answers to them. However, life sciences have made tremendous progress within the last couple of years. Every single day we accumulate more knowledge about the unsolved mysteries of our bodies and minds.

Our brains have amazing computational abilities. With ease they interpret the vast amount of sensory data flowing into them, and in fractions of seconds they react to the ever-changing world around us. Researchers around the globe are building machines that appear to have similar capabilities. Supercomputers have beaten the best human Backgammon[2], Chess[3], and even Jeopardy [4] players. The paradigms upon which they function seem, unfortunately, fundamentally different from how humans think and reason.

Deep Blue, the digital Chess master, can only play Chess. Watson, the digital Jeopardy champion, can only play Jeopardy. Both of them could never figure out by themselves how to play Chutes and Ladders. Something seems fundamentally broken in our most "intelligent" machines.

---

[1]Several text passages in this chapter are taken and modified from papers I have co-authored and my research plan.

[2]BKG 9.8, written in the late 1970s on a DEC PDP-10 was the first computer program to defeat a ruling world champion, Luigi Villa (7:1).

[3]The IBM supercomputer Deep Blue won on May 11^{th} 1997 against world champion Garry Kasparow in a six game match 3.5:2.5 (with 3 draw games).

[4]Watson is an AI system capable of finding the right questions to answers posed in natural language. It was built by IBM and won on February $14^{th} - 16^{th}$ 2011 against Bratt Rutter and Ken Jennings, two of the best Jeopardy players of all time.

The State of the Art. Existing computational models of cognitive processes can be subdivided in two broad classes: (*i*) top-down models like the mentioned machine-learning examples that aim to reproduce high-level abilities, without worrying whether the model is in any way similar to the implementation in our brains, and (*ii*) bottom-up models that focus on using biologically plausible modules that are based on established neurophysiological and neuroanatomical knowledge. The ultimate goal is to understand the detailed neural correlates of cognition – a goal that is currently out of reach.

This thesis takes some steps towards this ultimate goal: we present novel bottom-up models that are capable of adapting to a-priori unknown environments. Although we are not yet at the point where we can scale our models to become cognitive systems that solve complex real-world problems, we are convinced that our contribution is a step in the right direction.

A Short Tour Through this Thesis

In this thesis we present bottom-up models and learning procedures. They cannot (yet) play Backgammon or Chess, but they use novel strategies for how simple, brain-like systems can adapt to an a-priori unknown world around them. Once they have "understood" the structure of their environment they can use this learned knowledge to solve simple tasks like inference, cue-integration, biased and unbiased decision making, and signal restoration in a way that can also be observed in brains.

We hope that the network dynamics and learning strategies presented in this thesis will turn out to be adequate, though simplified, models of important computational primitives for cortical computation.

We are of course not the first to build biologically inspired networks performing brain-like tasks. In Chapter 2 we will lay out the scientific context of our own work and mention previous results that motivated and inspired us.

Part 1. In this part of this thesis we use what we call "classical competition models" (CCMs) as central building blocks. CCMs are systems with "soft winner-take-all" (WTA) dynamics [**DM07**] like the ones described by Amari [**Ama77**] or Heeger [**Hee92**].

In Chapter 3 we consider one of the key properties of our brains, the ability to notice and learn the relations between perceived entities [**CHK83, Pav60, RW72, WLHP68**]. It is believed that this is achieved by modifying the structure [**BK93, BC83, BC88**] and the dynamics [**KS61, SS01, RRMS10**] of biological neural networks, for example through the plasticity of synapses or other neural processes [**BBK06, Kan91, MGM00**].

We present a model that can learn the relationships between inputs in an unsupervised way (that is, without externally supplied error signals). In fact, our model is purely based on biologically motivated building blocks like population coding, Hebbian learning, and homeostatic activity regulation. After learning the relationship, our model can use the learned relation to improve its population code representations: the network will produce population codes for missing inputs based on supplied inputs (*inference*), will smooth noisy population codes (*denoising*), will adjust population codes to be more consistent with each other (*cue-integration*), and will choose between alternative population code representations when faced with inconsistent data (*decision making*). A key feature of our network is that its dynamics do not have to be modified from the outside in order to switch between these tasks, or even to re-learn the relationship when it changes.

In Chapter 4 we ask a more developmental question. It is known that inter-areal projections are often topographic in nature [**PKD$^+$06, EZ78**], meaning that the relative positions of the terminal axonal arbors in the target area are arranged similarly to the relative positions of the somas in the source area. However, the terminal arbors often overlap significantly, with a single arbor covering from 5% to 30% of the total target area [**KSBH94**]. The question is whether the synaptic connections might provide more precise topographic connectivity than one would assume just by examining the morphology and assuming random connectivity [**BS91**] within the axonal and dendritic arbor regions.

We find that the same combination of mechanisms we used in Chapter 3 is capable of sharpening inter-areal projections in a variety of network architectures. Furthermore, in networks with recurrently connected areas, these mechanisms result in sharpened back-projections being aligned with sharpened forward projections.

Although we use CCMs as a central building block in Chapters 3 and 4, we are not fully satisfied with some of their properties. We used them

because they are well known, standard models that offer the WTA dynamics we wanted to exploit for building the proposed systems. Unfortunately, CCMs are fairly rigid beasts that cannot easily be mapped onto the anatomy of cortical structures. This also means that every system based on CCMs inherits this problem and is also not easily mappable onto cortical anatomy. This is a problem insofar as the cortex is the structure these systems attempt to model.

As a logical consequence we asked ourselves how networks that are likely to exist in the cerebral cortex could create WTA dynamics. In the second part of this thesis we develop one possible answer.

Part 2. We start by using a very general, recurrent network motif that appears in similar forms in many places in the central nervous system of various species [**ASLB07, PMBA$^+$09, BDM04, BS91**]. The basic idea is that neural populations consist of excitatory and inhibitory neurons that, in their initial, 'tabula-rasa' like configuration are essentially randomly connected [**KSM05, PBM11**]. The canonical microcircuit [**BDM04**], for example, contains several places where this motif can be found.

In Chapter 5 we introduce "Recurrent Competitive Networks" (RCNs). We argue about their relevance as cortical models and analyze some of their rich dynamics. We then show in detail how RCNs respond to external input and point at similarities and differences to WTA dynamics.

The networks we discuss in this chapter consist of spiking neurons[5] with fairly detailed synaptic dynamics. Since it is surprisingly difficult to ascertain biologically plausible parameter ranges for leaky integrate-and-fire (LIF) neurons we include a literature review in Chapter 6. This chapter also contains a fairly detailed explanation of the LIF neuron model itself.

This in-depth look at spiking RCNs points to a common difficulty of spiking simulations: spike patterns of cells in a network tend to synchronize, eventually showing barely plausible, erratic activities. In Chapter 7 we tackle this problem by analyzing synchronization and desynchronization of spike events in RCNs. We find setups that can control the synchronization dynamics for a wide variety of input distributions and input strengths. A key insight in this chapter is that the variabilities of

[5]In this thesis we use leaky integrate-and-fire (LIF) neurons for all spiking simulations.

biological neurons that we saw in Chapter 6 is not affecting the obtained simulation results. Networks of "unreliable" components do even show improved stability to degraded input patterns and are therefore able to propagate information at least as reliable as networks that consist of 'neuronal monocultures'.

Spiking simulations such as the ones in Chapters 5 and 7 use biologically plausible neuron and synapse models, but they come with a tremendous drawback in terms of simulation complexity. Studying learning in large networks takes an exorbitant amount of simulation time (at least on the type of hardware that is naturally available to most scientists).

In Chapter 8 we discuss this misery and suggest possible ways to dodge the problem. Finding a suitable higher level of abstraction that makes simulations faster without affecting the properties of the modeled system is one way out. By introducing the Siegert neurons **[RBH+11, OB11, Bur06, Ric77, Sie51]** in Chapter 8 we prepare ourselves for precisely that: using a suitable higher level of abstraction for RCNs.

Part 3. In Chapter 9 we describe a learning system built with Siegert neurons. We show that an RCN is capable of learning the topology (the population code structure) of typical inputs that are fed to the network. In case the inputs are structured like one-dimensional population codes, a trained RCN will respond to further stimulation like a one-dimensional WTA network does. If we instead feed a two-dimensional population code to the very same network, the learned connections cause this network to behave like a two-dimensional WTA.

The initial question, of how networks that are likely to exist in the cerebral cortex could create WTA dynamics, is thereby finally answered in this chapter. But trained RCNs do not only offer an alternative implementation of CCMs, they also come with an important additional feature of being able to learn the topology, the encoding of their inputs. This is an important step towards systems that can adapt to complex, a-priori unknown environments.

We push this ansatz even further in Chapter 10, where we describe larger networks that couple trainable RCNs. These coupled RCN layers can exchange information about the input they receive via sparse excitatory connections. The network's task is, very much in the spirit of Chapter 3, to figure out and remember how these population-encoded input values relate to each other.

The differences from the system described in Chapter 3 are that (*i*) we use RCNs rather than CCMs for getting the desired WTA dynamics, and (*ii*) the neuron model we use is not an artificial sigmoidal threshold unit, but a rigorous abstraction from leaky integrate-and-fire neurons, the Siegert neurons, and most important (*iii*) the coupled RCNs are able to adapt to the input structure given by the a-priori unknown environment the system is thrown into.

We believe that these differences will turn out to be crucial modifications necessary for building large-scale models that will eventually solve complex real world learning tasks.The results collected in this thesis are a step towards our ultimate goal of building artificial neural systems that can adapt to dynamic environments.

CHAPTER 2

Background and Related Work[1]

Our work is inspired by (*i*) models which we refer to as *cortical field models*, (*ii*) *factor graphs* and other graphical models used to build inference systems, and (*iii*) a series of anatomical, physiological, and modeling work on *recurrent neural networks*. This chapter will look at each of these in more detail. All these results and models explore in different ways the question of how neural architecture relates to the sort of computation that we see evidence of in the brain. The results contained in this thesis do not follow directly from any particular one of these models, but do follow the intellectual tradition of all of them, including an emphasis on using large simulations as a window to gain insight into how theoretical principles may play out in the brain.

2.1. Cortical Field Models

In the following we use the term "cortical field" in a quite general way. It denotes a layered system with feed-forward and possibly also feed-back connections between these layers. Each layer consists of a large array of units, which may represent neurons, small constellations of neurons, or even cortical columns. Within a layer, we have lateral connections between neighboring units, and possibly also different classes of units such as excitatory and inhibitory cells, or simple cells and complex cells, or some other local division of labor. However, each layer is homogenous in the sense that this structure is the same throughout the layer.

Plastic cortical fields come with a learning rule that describes the nature of their plasticity. This learning rule ought to be biologically plausible in the sense that the learning that occurs at each synapse (or more generally, at each connection between units) must depend only on information that is locally available at that synapse; i.e., the activity of

[1]Several text passages in this chapter are taken and modified from papers I have co-authored and my research plan.

the presynaptic and postsynaptic units, and possibly a global signal corresponding to a neuromodulatory signal.

The attraction of cortical field models is that they show us how interesting non-homogeneous characteristics can arise from an initially homogeneous substrate. Below we give a list of models that inspired us most.

LISSOM: (laterally interconnected synergetically self-organizing map) is an approach by Miikulainen et al. [**MBCS05**] to construct an abstract model of the early visual system (up to V1) that exhibits more elements of the structure found in the visual cortex than the "simple" self-organizing maps of Kohonen [**Koh97**] do. The LISSOM model consists basically of a two-dimensional matrix of computational units, each corresponding to a vertical column in the cortex. These units are, on the one hand, laterally connected to other units in the surrounding area and receive input from the retina. The main ability of the LISSOM model is the generation of cortical maps that appear very similar to the ones found in primary visual cortex, in particular, orientation preference maps [**BZSF97**] or ocular dominance maps [**BG92**].

Suarez, Koch and Douglas: [**SKD95**] created a functioning model of direction selectivity in visual cortex based directly on the laminar anatomy. They modeled multiple layer networks consisting of integrate-and-fire-neurons and associative synaptic connections between stages. Their findings from the investigation of information retrieval suggest that recurrent circuits can support cortical information processing under certain conditions.

TRN: Dominey [**Dom06**] has developed the recurrent network model TRN (Temporal Recurrent Network) that is able to encode the interaction between serial order and temporal structure, similar to the Liquid State Machines of Maass [**Maa07**] (see also below). Based on the temporal organization of sentences the model is capable of discriminating different syntactic structures.

HMAX: was developed by Riesenhuber and Poggio [**RP99**]. It reflects the hierarchical organization of the visual cortex in a series of layers. These layers can be divided into two different types. The first performs a convolution operation on the input using a certain kernel, for instance, the 2D-Gabor-function. The second type of layer performs a soft maximum operation on its input in order to provide position invariance. These two types of layers are arranged

in an alternating order starting with a convolution layer. HMAX is good at position and scale invariant object recognition and has been applied to a number of problems.

TELOS: Yet another cortical model. TELOS, created by Brown, Bullock, and Grossberg [**BBG04**], is formulated according to several computational hypotheses which specify how strategy priming and action planning are dissociated from movement execution by assigning different functional properties to different layers in modeled cortex.

Haeusler and Maass: have explored the idea that the laminar architecture serves simply as a slightly better alternative to a completely random architecture [**HM07**], as measured by considering the information present in the emergent dynamics of a simulation.

LAMINART: [**Gro99**] extends Grossberg's well known Adaptive Resonance Theory (ART) model in a way that it provides clear functional roles for its layers for purposes of visual perception, and it suggests that similar functional roles may be at work in sensory and cognitive processing.

Treves: showed in [**Tre03**], that the use of laminar model structures can allow networks to represent positional information through cortical maps while simultaneously representing identity information through firing patterns.

2.2. Factor Graphs

Factor graphs are a well-known formalism encompassing many models that were independently invented in the fields of artificial intelligence, signal processing, coding theory, and statistical mechanics [**AM00, FJ01, KFL01, Pea88, YFW05**]. The appeal of factor graphs, from our point of view, is that they have several properties which point to the possibility of a neurobiologically plausible implementation.

- Information is stored in a distributed manner, integrated into the computing elements.
- Computations are performed locally using locally available information.
- Learning can occur via local mechanisms based on locally available information.

None of these features were intentionally built into the factor graph model. Factor graphs in their various forms were constructed independently by engineers in several fields simply to help solve complex probabilistic problems involving many interrelated correlations. They chose their models simply as the best existing way to solve the problem. Most applications do not involve a learning stage.

Recently, factor graphs have started to attract the attention of some neuroinformaticists. Cook and Bruck have shown how factor graphs can be used not only as a probabilistic representation of complex data, but also for nonprobabilistic purposes such as real-time robotic control, and solving the inverse/forward model problem [**CB04**]. Thus, there is hope that models of this sort can close the loop from analysis of sensory input to production of motor output, forming a complete system out of a uniform substrate. These ideas are still in their infancy, but this thesis contains attempts to implement factor graph like inference mechanisms using neuronal modules.

It is interesting that there also exist papers outside the factor graph literature which find neural implementations of computations that in our view closely resemble the computation performed by a factor in a factor graph. This variety of neurally implemented factor-like models, such as for example [**WW07**] or [**DLP01**], reaffirms our conviction that there is enough flexibility in this approach to implement neural factors which can be combined to form practical factorizations of complex real-world problems.

2.3. Recurrent Neural Networks

The term *Recurrent Neural Network* is very general and fits most neurobiologically relevant networks. In this thesis it will usually refer to a specific line of research on cortical models of recurrently connected excitatory and inhibitory neurons that exist in very similar forms in many places in the central nervous system of various species [**ASLB07, PMBA$^+$09, BDM04, BS91**]. The canonical microcircuit [**BDM04**], for example, contains several places where such structures can be found.

Recurrent networks like the ones we use in this thesis are relatively common. It is therefore not very surprising that a plethora of theoretical work on such networks can be found in the literature [**HRB11, RRWF10, VA09, KRA08, BSF07, VA05, DGFM03,**

BW03, Bru00, AB97]. The following paragraphs introduce some key results that inspired our working and thinking.

Roughly 15 years ago, Amit and Brunel [**AB97**] started to look at networks like the one shown in Figure 5.1. This and subsequent work contains detailed mathematical analysis of dynamics in such networks. [**Bru00, HRB11**]. The input to their networks is assumed to be Poisson distributed, which is not necessarily the case in real world scenarios [**BMS07, DCM09, DCM11**]. Each population of cells they use in their studies consists of a monoculture of cells, where each cell is a perfect copy of every other.

A large body of work is available on synfire chains (or synfire braids) [**KRA10, DC06, Izh06, DGA99, Abe82**]. These models investigate how small, synchronously active cell populations can transmit activity throughout large networks and how interactions between such chains has the potential to perform neural computations. Since synchrony is essential for this kind of model, conditions for synchronous spike- vs. asynchronous rate transmission were investigated [**KRA08**]. One observation is that the spiking activity in recurrent networks tends to synchronize as it propagates through a network, indicating that asynchronous spiking is not always easy to achieve in recurrent feed-forward networks of spiking neurons [**KRA08, Rey03**].

Although these works offer intriguingly intuitive ideas of how synchrony might be a fundamental concept of neural computation, there is experimental evidence that suggests the opposite. It appears as if nearby neurons in cortical networks might even be actively de-correlated [**EBK$^+$10**].

Gutkin et al. [**GLC$^+$01**] showed that in a recurrent network model capable of sustaining activity, asynchronous firing is necessary to sustain it as long as the network is in the memory state. The sustained activity can in fact be switched off by inducing the network to synchronize.

It seems that science has not yet identified the fundamental computational principles used by cortical structures in our brains. Many promising and conflicting ideas do exist, but more ideas will be needed to complete the picture.

Part 1

Learning in Classical Competition Models

CHAPTER 3

Unsupervised Learning of Relations[1]

3.1. Introduction

One of the key properties of the brain is the ability to notice and learn relationships between inputs in an unsupervised manner [**CHK83, Pav60, RW72, WLHP68**]. It is believed that this is achieved by modifying the structure [**BK93, BC83, BC88**] and the dynamics [**KS61, SS01, RRMS10**] of biological neural networks, for example through the plasticity of synapses or other neural processes [**BBK06, Kan91, MGM00**].

Phenomenologically, the ability of brains to discover relationships between otherwise independent events has been known since the pioneering work by Pavlov [**Pav60**]. His work on dogs showed that a neutral stimulus (the ringing of a bell) can be induced to elicit an associated reaction (production of saliva) by ringing a bell every time the dog gets food. After training, the link between the bell and salivation may be due to the food representation being activated by the bell representation.

In the following decades, it became clear that the ability to learn relations between different sensory inputs is in fact omnipresent in our brains. As an example of an *inference* task, when we hear a sound, we can use the audio input to estimate the visual location of the corresponding visual input. This process is continually maintained by learning mechanisms: if we wear prism glasses that shift the visual input, it is possible to re-learn the correspondence. Or, given uncertain visual and audio cues about the location of a stimulus, we can combine these cues (*cue-integration*) to get a better estimate of the location. If visual and audio cues differ so much as to be inconsistent, for example due to light or sound being reflected so as to appear to have a different source,

[1]The content of this chapter is published in [**CJK10**].

then we simply base our position estimate on the stronger of the two competing inputs (*decision*).

Biological data shows that neural populations, regions and areas encode specific sensory, motor, and cognitive modalities (see e.g. [**FvE91, SA01**] and contained references). Connected regions exchange signals and thus influence their mutual activity [**FvE91**], and simulations have also exhibited such interactions in networks with hand-crafted connectivity [**SS01, SA01, PS97**]. Simulations such as these have shown how inference, denoising, cue-integration, or decision tasks can be performed on input received from different visual, motor, or other sources, for complex problems like coordinate transformations. However, exhibiting these abilities in networks that learn the relationships, rather than using hand-crafted weights, has remained a challenge.

Given the presence of such abilities in the brain, the question that immediately arises is *how* the brain implements them. A major step in this regard was achieved by Zipser and Andersen [**ZA88**], who trained an artificial neural network with simulated biological data using the backpropagation algorithm [**RHW86**]. In their network, hidden nodes developed gain field properties [**ZA88**]. While their result shows that neural networks are able to learn such tasks in principle, the learning strategy they used seems unlikely to be the one used in our brains: the back-propagation algorithm is a supervised learning scheme, using an externally generated error signal, and it is generally considered to be biologically implausible [**Cri89, ZR93**].

Our goal is to exhibit the ability to learn arbitrary relationships using biologically plausible learning. We present a model that can learn the relationships between inputs in an *unsupervised* way (that is, without externally supplied error signals). In fact, our model is purely based on biologically motivated building blocks like population coding, Hebbian learning, and homeostatic activity regulation. After learning the relationship, our model can use the learned relation to improve its population code representations: the network will produce population codes for missing inputs based on supplied inputs (*inference*), will smooth noisy population codes (*denoising*), will adjust population codes to be more consistent with each other (*cue-integration*), and will choose between alternative population code representations when faced with inconsistent data (*decision*). A key feature of our network is that its dynamics do not have to be modified from outside in order to switch between these tasks, or even to re-learn a relationship when it changes.

3.2. The Network and Its Dynamics

In this paper we consider the following network (see Figure 3.1) to demonstrate how relations between two sets of parameters X and Y can be learned. The network consists of two populations, A and B, consisting of n rate coded units each.

3.2.1. The Network. The units in A get input from an external source X by point-to-point connections, i.e., each unit in A receives input from exactly one unit in X and each unit in X sends input to exactly one unit in A. Similarly, a second input Y, is connected to B by point-to-point connections. X and Y are supposed to encode one single scalar value each. To realize this encoding we use what is known as population coding, see e.g. [**DLP01**]. Intuitively, this means that each unit in X has one preferred value and that its firing rate depends on how close its preferred value is to the actual value. In Figure 3.1 we illustrated this encoding by representing X and Y by two (noisy) population codes.

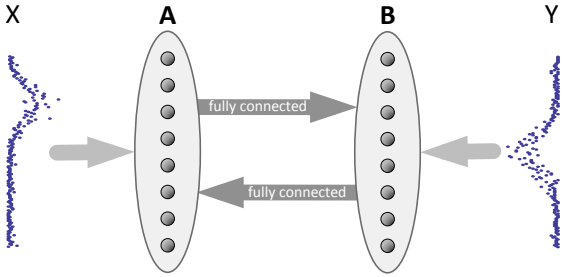


FIGURE 3.1. Projection diagram of the sample network discussed in the text: two populations with bidirectional connectivity. Labeled ellipses represent populations of neural nodes. Dark gray arrows depict directed, full connectivity, light gray arrows indicate point to point connections used to feed population-coded input into the network. Blue dots show one possible input of this kind.

The units within each of the populations A and B are laterally interconnected such that each population is effectively a soft winner-take-all circuit [**DM07**]. The connection weight $w_{i,j}$ between units i and j is

defined as:

$$w_{i,j} = \gamma \cdot e^{-\frac{1}{2}(d(i,j)/\sigma)^2} - \delta\,. \tag{3.1}$$

The distance d between i and j is $d(i,j) = \min\{|i-j|, n-|i-j|\}$. In order to avoid boundary effects we let the distance measure wrap around.

Besides being laterally connected, the units in A and B are also interconnected. Effectively we connect each unit $i \in A$ to all units $j \in B$ and vice versa. The initial connection weights $w_{i,j}$ are set to values chosen randomly in $[0,1]$.

Learning the relations between the inputs X and Y is done by adapting the connections between the populations A and B using a Hebbian learning rule [**Heb49**].

3.2.2. The Dynamics. We simulate our network over discrete time steps. At time t the rate-coded units in A and B each have a real-valued activity level a, which we denote with a superscript as a^t. At each time step t each unit j updates its activity level a_j^t. This update is influenced by (*i*) the activities of the neurons in the same populations (via the lateral connections), (*ii*) the activities of the units in the other population (via the connections between A and B), and (*iii*) a homeostatic activity regulation term h_j^t (used to keep the activity level of each unit roughly constant over time).

We explain the details of the update below. Here we just outline the interplay between the main ingredients. The lateral connections implement soft winner-take-all dynamics (WTA) [**DM07**]. Essentially, they are used to "clean-up" noisy or multi-modal input. The weights $w_{i,j}^t$ between the populations A and B are updated by a Hebbian learning (HL) scheme, eventually encoding the learned relationship. The homeostatic activity regulation (HAR) [**TN04**] forces units to regulate themselves so that each unit is active roughly a given proportion of the time. This makes sure that every unit is used, and that each unit is used in moderation.

It is worth noting that the presented components work on quite different time scales. The WTA dynamics operate on a short time scale, allowing the network to converge quickly. HAR and HL operate on a much longer time scale, averaging over a much larger sample of inputs. A sketch of how Hebbian learning (HL), soft winner-take-all (WTA) and

homeostatic activity regulation (HAR) play together is illustrated in Figure 3.2.

Hebbian Learning. The update of the weights $w_{i,j}^t$ depends on (i) the activities a_i^t and a_j^t of units i and j at time t, and (ii) two global parameters α_l and α_d. The Hebbian learning rate α_l regulates the speed at which connections get learned and was in our simulations usually set to the same value as α_d, the Hebbian decay rate. The weights are updated according to:

$$w_{i,j}^{t+1} = (1 - \alpha_d) \cdot w_{i,j}^t + \alpha_l \cdot a_i^t \cdot a_j^t \,. \tag{3.2}$$

To speed up the running time of simulations it suffices to do these updates only after the WTA has converged.

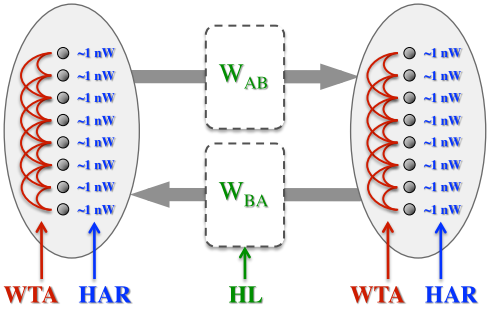


FIGURE 3.2. The presented mechanism is a combination of three strategies. Synaptic connections between areas are controlled by Hebbian learning (HL). Local connections within an area support soft winner-take-all (WTA) dynamics, so nearby units within an area exhibit similar activity patterns. Homeostatic activity regulation (HAR) within each unit modulates the Hebbian learning so that a unit does not become permanently active or inactive, but maintains a desired average activity level.

Homeostatic Activity Regulation. We use the following update formula for the homeostatic activity terms:

$$h_j^t = -c \cdot (\bar{a}_j^t - a_{\text{target}}) \,, \tag{3.3}$$

where c is a scaling constant, a_{target} sets the desired activity level, and $\bar{a}_j^t$ is a running average of the activity of unit j, defined by

$$\bar{a}_j^t = (1 - \omega)\bar{a}_j^{t-1} + \omega a_j^t \tag{3.4}$$

where ω is the inverse time constant of the averaging.

Neural Units and Update Dynamics. At each discrete time step t each unit j updates its activity level a_j^t. To compute it we first take the weighted sum over the activity levels of all units connected to unit j. This includes both the lateral connectivity within the population as well as the connections coming from other populations. This sum is corrected by the homeostatic activity regulation term h_j^t. Finally we apply a non-linear function θ that restricts the activity level to the range $[0, 1]$. Formally the update rule is defined as

$$a_j^{t+1} = \theta(h_j^t + \sum_{i \in \Gamma_j} w_{i,j}^t \cdot a_i^t)\,, \tag{3.5}$$

where Γ_j is the set of units connected to unit j, and

$$\theta(x) = \frac{1}{1 + e^{-m(x-s)}} \tag{3.6}$$

and m and s are parameters that determine the slope and the shift of $\theta(x)$.

3.3. Results

In the following we present our experimental results. Note that the network dynamics introduced in the previous section remains unchanged throughout all experiments that we present. In order to switch from one task to another we only change the input fed to the network.

3.3.1. Learning and Re-learning. In order to feed interpretable input we have set a preferred stimulus p_i for each node i in A and B. To encode the value v in X (or Y) we set the input x_i (y_i) for node i in A (B) according to:

$$x_i(v) = C \cdot e^{-(v-p_i)^2/(2\sigma^2)}\,. \tag{3.7}$$

This enables us to feed arbitrary scalar values to populations A and B. If these values satisfy any functional relation, the network will learn the relationship hidden in a sequence of input pairs. Note that the weights between populations A and B are constantly changing over time. If after a certain relationship was learned the input changes and a different relation is presented, the weights will change to reflect the new relation. Figure 3.3 shows how the weight matrices W_{AB} and W_{BA} change in the course of learning and re-learning.

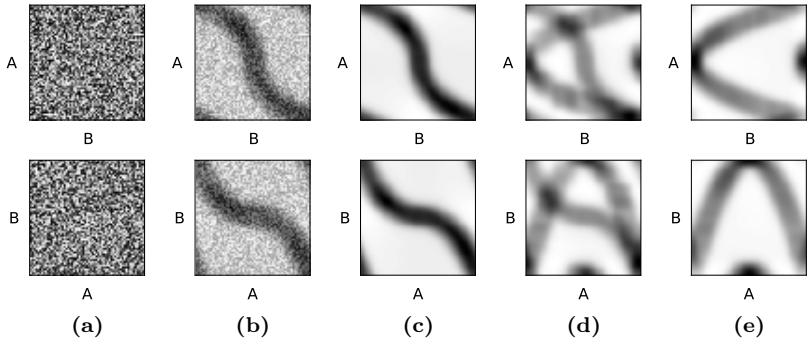


FIGURE 3.3. The time course of learning and relearning in the sample network. Each plotted subfigure shows a snapshot of the connection weights W_{AB} (top row) and W_{BA} (bottom row) for different times during learning and relearning. The weights are color coded (black for strong, white for weak connections). **(a)** Initial random weights. **(b)** Weights during learning. **(c)** Weights after the relation $y = x^3$ was learned. **(d)** Weights during relearning. **(e)** Weights after the relation $y = x^2$ was learned.

3.3.2. Inference Tasks. After the network has learned a relation we can then also omit one of the inputs and infer the other value. This is done as follows. We only feed input in X (or in Y) and let the network converge. After convergence one can use the activities in A and B to compute the population vector **[GKCM82]** giving us the values v_A and v_B encoded by A and B.

Figure 3.4 shows the result of such inference tasks. We tested the inference accuracy by encoding all values $v_i \in \{p_i | i \in A\}$ in X (respectively all values $v_j \in \{p_j | j \in B\}$ in Y) and observing the values v_B (respectively v_A) computed by the network.

3.3.3. Denoising and Cue-Integration Tasks. In all of the following examples we add some noise on top of the activations computed with Equation (3.7). Figure 3.5(a) shows how the network performs inference with noisy input.

In addition to such noisy input signals our network can also cope with noisy values v_X and v_Y. Figure 3.5(b) is an example for the case when the inputs in X and Y are not in line with the learned relation R. The

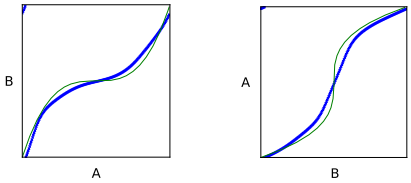


FIGURE 3.4. Simple inference in sample network after the network has learned the relation $y = x^3$ (green thin line in both plots). **(a)** shows the results of the inference tasks (thick blue line) for a set of population codes fed to A (horizontal axis). **(b)** like (a) but for the opposite inference direction.

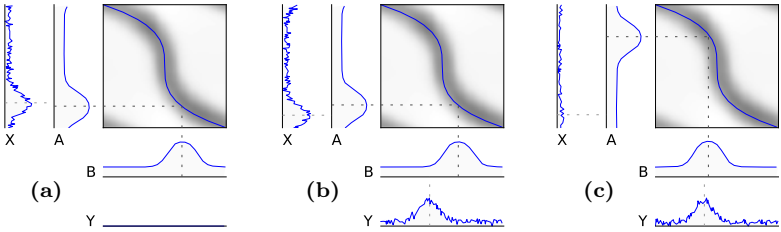


FIGURE 3.5. Inference and denoising. In each subfigure, the inputs are given in X and Y, the results can be seen in A and B. **(a)** An example of inference from X to B which shows also the denoising properties of the network with respect to noise in the firing rates of the units. **(b)** When two inputs are presented which are inconsistent with the learned relation the network shifts both peaks until their positions are in accordance with the relation. **(c)** The same as (b) but with unequal reliability of the inputs (unequal input strength); note that the larger (more reliable) peak is much less shifted than in (b).

network settles in a state where the computed values v_A and v_B are again consistent with R. Figure 3.5(c) shows the same experiment but with different input strengths in X and Y. Note that the population receiving the stronger input gets significantly less shifted towards a place consistent with R than the other one.

Note that the soft winner-take-all implemented in our populations A and B is the reason why the described phenomena work.

Clearly, a cue-integration task (like in Figure 3.5 (b)) gets more and more difficult (and unreliable) depending on "how much" the value fed into Y differs from the "true" value that is consistent with the input in X. Eventually, if this difference gets too large, then the system will stop finding a compromise between these two values and instead will start to neglect one of the inputs. That is, the network will decide between the two values.

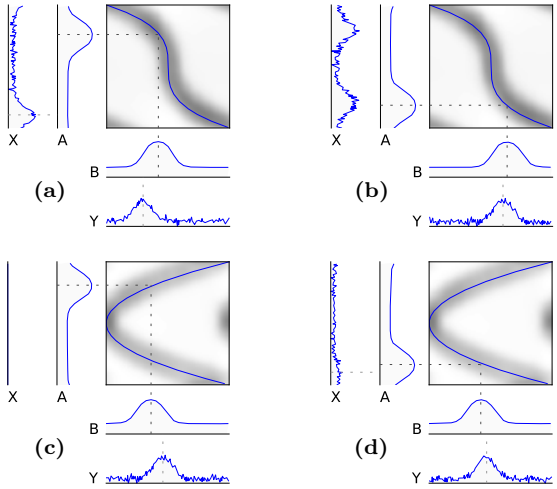


FIGURE 3.6. Decision tasks. **(a)** When the peaks of the inputs are not close to the learned relationship the network uses one of the inputs and infers the other one. **(b)** When in X there are two contradicting inputs present, while one is being supported by the input in Y, the network decides for that combination of peaks. **(c)** In the case of a non-invertible function like $y = x^2$ there exist two possible peak positions and the system decides for one of the two. **(d)** The same as (c) but the network's decision is biased by a very small input fed to X.

3.3.4. Decision Tasks. As indicated at the end of the previous section, the network can be forced to decide whether to follow input X

or input Y. Figure 3.6 shows the input and the settled state for four decisions being performed by the network.

Given inputs of similar strength the noise determines how the network will decide. If the inputs are of equal strength the network will essentially decide on one of them "randomly", meaning that small artifacts from the learning history will be responsible for the decision.

Figure 3.6(b) illustrates another, more complicated decision task. In this example the input in X actually contains two peaks. If the second input relates to one of these two peaks (with respect to R), the network will reinforce this peak and settle in a state consistent with R.

If the learned relation R corresponds to a non-invertible function (like $y = x^2$ for x in $[-1, 1]$) then, clearly, an input in Y may be in correspondence with more than one consistent X-value. The network will then have to pick one of the possible solutions. This example is illustrated in Figure 3.6(c). In addition, Figure 3.6(d) illustrates that already a seemingly small "noise" in the input in X suffices to move the generated value to the one that has a higher consistency with the input.

3.4. Discussion

In this chapter we showed that it is possible to set up the dynamics of a simple network in such a way that it can learn the relation between two inputs X and Y. After learning, that is, after presentation of sufficiently many related input pairs, the network is then able (*i*) to *infer* missing input, (*ii*) to clean up noisy population codes (*denoising*), (*iii*) to mediate between slightly conflicting inputs *cue-integration*, and (*iv*) to *decide* between strongly conflicting inputs. If one continues to present strongly conflicting inputs, the system will gradually change and eventually learn the new relation.

The building blocks of our network, namely population coding, soft winner-take-all, Hebbian learning, and homeostatic activity regulation, are all biologically well motivated.

A aim for future research is to learn higher order relations between more than two input signals. To achieve this it will be necessary to replace the effectively one-dimensional populations used in our network by more complex recurrent networks capable of encoding these higher order relationships. Indeed, the internal connectivity of the areas, reflecting the topology of the input space, would ideally be learned based on the observed inputs themselves (a system like this will be presented

in Chapter 9). This would allow both higher dimensional transformations (such as those related to gain fields [**SS01**]) and more abstract relationships to be learned with the same mechanisms.

CHAPTER 4

Competition During Development: Sharp Learning of Neural Projections[1]

4.1. Introduction

A recurring theme of inter-areal projections is that they are topographic in nature [**PKD+06, EZ78**], meaning that the relative positions of the terminal axonal arbors in the target area are arranged similarly to the relative positions of the somas in the source area. However, the terminal arbors often overlap significantly, with a single arbor covering from 5% to 30% of the total target area [**KSBH94**].

A natural question is whether the synaptic connections might provide more precise topographic connectivity than one would assume just by examining the morphology and assuming random connectivity [**BS91**] within the axonal and dendritic arbor regions.

Note that a precise projection does not necessarily imply small arbors. Even if non-precise connections are pruned during development, the remaining, precise synapses will still be distributed throughout the area where the original axonal arbor overlapped with dendritic arbors of target cells, as in Fig. 4.1(a,b). Thus the morphology alone cannot indicate whether such a sharpening process has occurred or not, and current anatomical knowledge does not yet include sufficient information on synaptic specificity of inter-areal projections [**BDM04, OF05**], leaving the question open.

[1]The content of this chapter is published in [**CJK11**].

The biologically plausible mechanisms that we use are Hebbian learning at synapses, continuous winner-take-all circuitry within areas, and homeostatic long-term activity regulation within neurons, as shown in

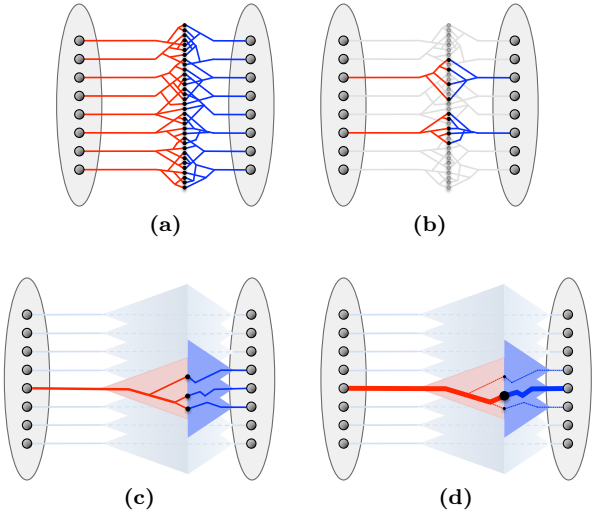


FIGURE 4.1. **(a)** A precise projection between two areas (grey ellipses) without small arbors. The synapses (black dots) can be distributed throughout the area where the projecting cell's axonal arbor (left, red) overlaps with the dendritic arbor (right, blue) of the target cell. Although the arbors have a chaotic and unfocused morphology, the projection is in fact perfectly topographic, one-to-one connectivity. **(b)** The same picture with most arbors grayed out. Here it is easier to see how the synaptic connections are providing a perfectly precise projection. **(c)** When one area projects to another, the terminal axonal and dendritic arbors (symbolized as shaded triangles) allow each projecting neuron to reach a range of targets (three shown). **(d)** Learning mechanisms can effectively sharpen the projection by strengthening some synapses and weakening others. This is symbolized here by showing the connection of the most-aligned units as strengthened, while other connections are weakened, yielding the connectivity of (a). For visual clarity, these diagrams (a)-(d) are vast simplifications of real arbors, which contain thousands of synapses in three dimensions, often centered around the target soma. In reality a projection would not have to be one-to-one to be considered precise, but it would need to use synaptic specificity to prefer localized targets.

Fig. 4.4. We find that this combination of mechanisms, which we refer to collectively as *sharp learning*, is capable of sharpening inter-areal projections in a variety of network architectures, such as those in Fig. 4.2. Furthermore, in networks with recurrently connected areas, sharp learning results in sharpened back projections being aligned with sharpened forward projections, as shown in Fig. 4.3(c,d).

Over the last decades, there have been countless models examining the training of weights between layers in a network. Our results are most closely related to the pioneering work of Willshaw and Malsburg, who modeled the development of unidirectional topographic retino-tectal projections in the frog [**WvdM76**].

4.2. Methods

Sharp learning takes place in the context of interacting groups of units that we refer to as *populations* or *areas*. The large-scale architecture of a network lies in the projections between these populations. These projections can be shown in a *projection diagram*, such as those shown in Fig. 4.2. One well-known projection diagram is that of Felleman and van Essen, showing the connectivity between cortical areas in the visual pathway of the macaque [**FvE91**].

The populations are composed internally of *units*, which can be considered as corresponding either to an individual neuron, to a small neural microcircuit in the cortex [**BDM04**], or to a tightly connected group of cells such as a cortical microcolumn [**Mou57**].

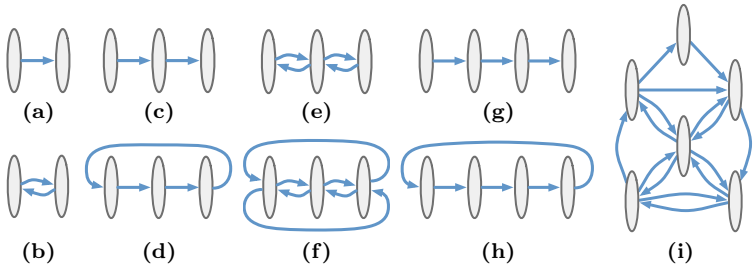


FIGURE 4.2. Examples of inter-areal architectures where sharp learning is successful, including feed-forward paths and cycles **(a,c,g,d,h)**, bidirectional paths and cycles **(e,b,f)**, and an arbitrary complex structure **(i)**.

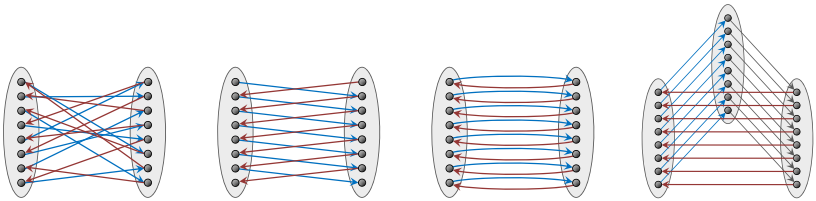


FIGURE 4.3. Various types of connections. **(a)** Shows disordered connectivity, as would be expected with randomly connected synapses. Such randomness is often assumed when considering connectivity on a scale smaller than a dendritic arbor [**BS91, BDM04**]. **(b)** Shows skewed connectivity, as would be almost inevitable from a developmental program of chemotaxic axon growth attempting to form reciprocal connections. **(c)** Shows reciprocal connectivity, which is achieved by sharp learning. **(d)** Shows how this idea can be generalized to cycles of length three or more. Again, aligned connectivity is shown, as achieved by sharp learning. The arrows in each diagram represent the strongest connection. Nearby connections (not shown) are also present but weaker.

Sharp learning is a combination of three strategies, as shown in Fig. 4.4 and Fig. 4.5. Synaptic connections between areas are controlled by Hebbian learning (HL), so that the weights reflect the correlation of typical network activity [**Heb49**]. Local connections within an area (*lateral connections*) support continuous winner-take-all (WTA) dynamics [**DM07**], so neighboring units within an area exhibit similar activity patterns and noisy input is smoothed. Homeostatic activity regulation (HAR) within each unit modulates the Hebbian learning so that a unit does not become permanently active or inactive, but maintains a desired average activity level [**TN04**]. This makes sure that every unit is used, and that each unit is used in moderation.

It is worth noting that the presented components work on quite different time scales. The WTA dynamics operate on a short time scale, allowing the network to converge quickly. HAR and HL operate on a longer time scale, averaging over many inputs.

Winner-Take-All. The units within each of the populations are laterally interconnected so that each population is effectively a continuous winner-take-all circuit [**DM07**], meaning that the dynamics lead to a localized region of activity, similar to the encoding of a value by a population code [**GKCM82**]. The connection weight $w_{i,j}$ between units i and j is defined as

$$w_{i,j} = \gamma \cdot e^{-\frac{1}{2}(d(i,j)/\sigma)^2} - \delta\,, \tag{4.1}$$

where $d(i,j) = \min\{|i-j|, n-|i-j|\}$ gives the distance d between i and j, with n being the number of units in the population. In order to avoid boundary effects we let the distance measure wrap around. The parameters γ, σ, and δ specify the amplitude, the width, and the vertical displacement (i.e. the amount of lateral inhibition) of the Gaussian shape of the connection weights profile.

Hebbian Learning. The update of the weights $w^t_{i,j}$ depends on (i) the activities a^t_i and a^t_j of units i and j at time t (incremented in the outer loop of Fig. 4.5), and (ii) two global parameters α_l and α_d. The Hebbian learning rate α_l regulates the speed at which connections get learned and is here usually set to a value smaller than α_d, the weight decay rate. The weights are updated according to:

$$w^{t+1}_{i,j} = (1-\alpha_d)\cdot w^t_{i,j} + \alpha_l \cdot a^t_i \cdot a^t_j\,. \tag{4.2}$$

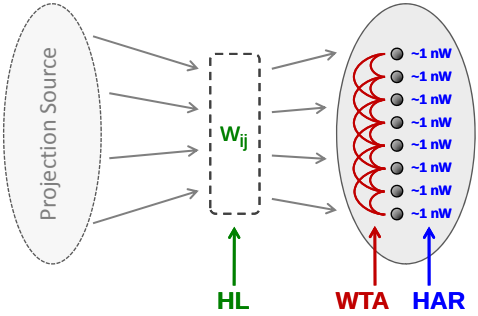
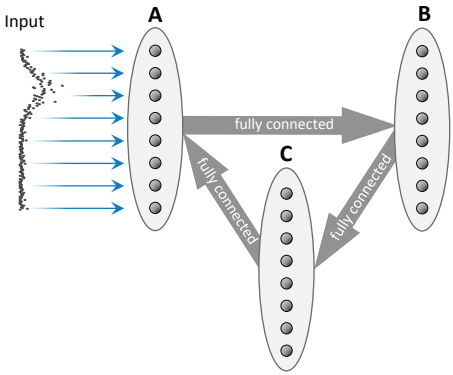


FIGURE 4.4. Components of sharp learning: Hebbian learning (HL) between areas, winner-take-all (WTA) within areas, and homeostatic activity regulation (HAR) within units.

We perform Hebbian learning only on inter-areal weights. To speed up the running time of simulations it suffices to do these updates only after the WTA converges.

Homeostatic Activity Regulation. We use the following update formula for the homeostatic activity terms:

$$h_j^t = -c \cdot (\bar{a}_j^t - a_{\text{target}}) \,, \tag{4.3}$$



Sharp Learning Algorithm

1: initialize inter-areal weights randomly
2: **loop**
3: initialize units with random activity
4: add input activity to population **A**
5: draw and feed noisy input to **A**
6: **repeat**
7: do WTA update (eq. 4.1)
8: **until** change in unit activities $< \epsilon$
9: do HL (eq. 4.2) and HAR (eq. 4.3)
10: **end loop**

FIGURE 4.5. The main loops in the sharp learning algorithm. The three-population ring architecture of Fig. 4.2(d) is shown, but the procedure is the same for all architectures. If lines 8 and 9 are swapped, then we refer to the modified algorithm as the "continuous learning" form of the algorithm.

where c is a scaling constant, a_{target} sets the desired activity level, and $\bar{a}_j^t$ is a running average of the activity of unit j, defined by

$$\bar{a}_j^t = (1-\omega)\bar{a}_j^{t-1} + \omega a_j^t \tag{4.4}$$

where ω is the inverse time constant of the averaging.

Neural Units and Update Dynamics. At each discrete time step τ in the inner loop of Fig. 4.5, we update the activity level a_j^τ of each unit j. To do this, we first sum the activity levels of all units connected to unit j, weighted by their connection strengths. This sum includes both the lateral connectivity within the population as well as the connections coming from other populations. This sum is regularized by the homeostatic activity regulation term h_j^t. Finally we apply a non-linear function θ that restricts the activity level to the range $[0, 1]$. This yields

$$a_j^{\tau+1} = \theta(h_j^t + \sum_{i \in \Gamma_j^{\text{in}}} w_{i,j}^t \cdot a_i^\tau)\,, \tag{4.5}$$

where Γ_j^{in} is the set of units connected to unit j, and θ is a logistic function

$$\theta(x) = \frac{1}{1+e^{-m(x-s)}} \tag{4.6}$$

parameterized by m and s.

Note that the time τ in Equation 4.5 refers to iterations of the inner loop (lines 6-8 of Fig. 4.5), while the time t in Equations 4.2-4.5 refers to iterations of the outer loop (lines 2-10 of Fig. 4.5).

4.3. Results

We show in Fig. 4.6 the results of sharp learning applied to the network of Fig. 4.2(d) as described in Fig. 4.5. Below each connection matrix shown in Fig. 4.6 is a diagram of the projection represented by that matrix, showing an arrow to the strongest target for each projecting unit. As described above, the populations in the simulation use a wrap-around topology for the lateral connectivity (the continuous winner-take-all), which is the only place we induce any topology into the network. Using a wrap-around topology allows an arbitrary displacement to arise in each projection, while still being precisely topographic. One can also see that the second and third matrices, once sharp learning has progressed, invert the ordering of the units within the population. The apparent discontinuities in the pattern of paths shown in Fig. 4.6(d) are in fact continuous, due to the wrap-around nature of the populations.

In Table 4.1 we compare the results of sharp learning for the network architectures shown in Fig. 4.2, with each population containing 200 units. The q-values in the table give the mean squared error compared

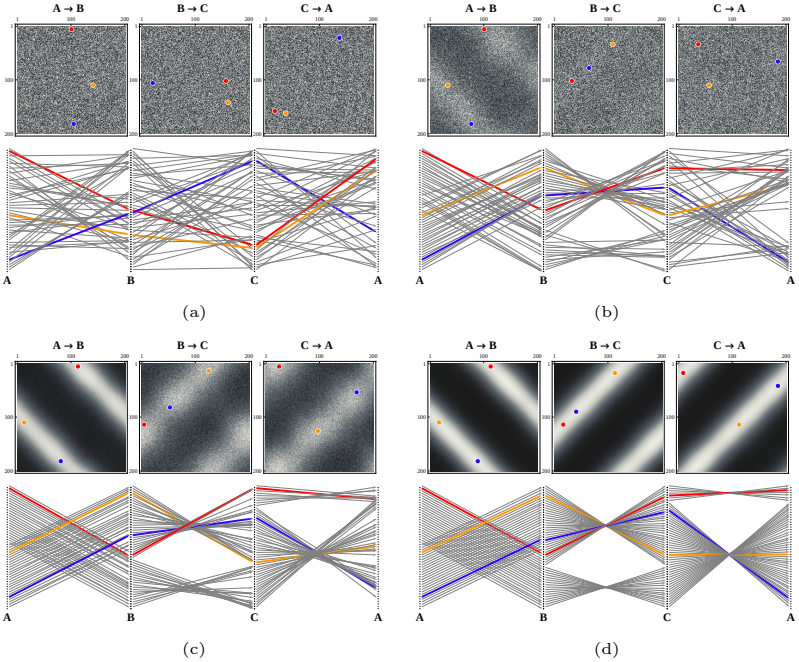


FIGURE 4.6. The weight matrices of the example network of Fig. 4.5. **(a)** shows the random initial state of the network, **(b)** and **(c)** show the state of the network during learning, and **(d)** shows the state after learning has converged. The matrices are shown with low values dark and high values white. Below each matrix is a mapping showing which target element each source element is most strongly connected to. In (a-d), the first matrix shows the weights from population A (row) to population B (column), the second from B to C, and the third from C to A. The goal of our algorithm, as reached in (d), is that neighbors within a source population should project to neighboring destinations, and the circular path starting from any unit should come back to that unit. Three examples of such paths are highlighted in different colors.

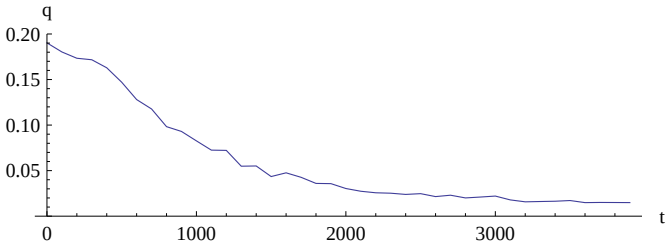


FIGURE 4.7. Evolution of quality value q for the network of Fig. 4.5. The q-values give the mean squared error compared to a perfect topographic mapping between all connected populations of a network.

to a perfectly topographic mapping between all connected populations of a network.

Fig. 4.7 shows how the q-value evolves over the course of sharpening the projections in a network, for the network shown in Fig. 4.5. Other networks, even with layers of unequal size, behave comparably (data not shown).

TABLE 4.1. Quality of sharp learning for the network architectures shown in Fig. 4.2. The quality q of a result is the root-mean-squared error in the position of the activity in each layer of the network, averaged over all possible positions of the peak of the input acitvity. The position of the activity in a layer is determined by considering the position of best fit of a Gaussian kernel, measured such that the size of the entire layer is 1. Due to the wrap around topology of the populations, the error is always between 0 and 0.5.

Networks	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)
Quality (q)	0.006	0.013	0.005	0.012	0.014	0.011	0.010	0.023	0.025

4.4. Discussion

We have shown that sharp learning is able to effectively sharpen inter-areal projections in a variety of circumstances, using a combination of biologically plausible mechanisms.

To further confirm the biological plausibility of sharp learning it will be necessary to investigate the robustness of sharp learning with respect

to inhomogeneities in the lateral connectivity within an area, as well as with respect to inhomogeneities in the homeostatic activity regulation and other parameters.

Another step towards biological plausibility would be to replace the mean-rate nodes used in our simulations by spiking units. In Part 2 we will discuss a model that does precisely this. Part 3, finally, will use a approximation introduced in Part 2 to simulate an adaptive spiking network in a rate-based setup. An entirely spiking implementation would require the transformation of the learning rules into a spike based equivalent.

We treat sharp learning here as a developmental process. We have also shown in Chapter 3 that a very similar procedure can learn data relationships fed to two populations.

Since sharp learning can (*i*) help to create precise topographic connections, and (*ii*) subsequently be used to learn relationships by simply observing input fed to the network, we believe that sharp learning is a capable model of learning whose power has only started to be explored.

Part 2

Recurrent Competitive Networks (RCNs)

CHAPTER 5

Introduction to Recurrent Competitive Networks (RCNs)

5.1. RCN Basics

Recurrent networks, such as the one shown in Figure 5.1, can be found in different places in the central nervous system of many species. It is therefore not very surprising that a plethora of investigations on such networks can be found in the literature [**HRB11, RRWF10, VA09, KRA08, BSF07, VA05, DGFM03, BW03, Bru00, AB97**].

Unfortunately we do not yet understand them well enough to be able to build artificial networks with brain-like data processing abilities. A key problem is that these systems are highly self-referential (recurrent) and cannot easily be tackled analytically. Although a mathematical description of the network dynamics is often possible using coupled differential equations, finding closed-form solutions for these is generally impossible.

The networks under investigation are very general and exist in very similar forms in many places in the CNS of various species [**ASLB07, PMBA$^+$09, BDM04, BS91**]. The canonical microcircuit [**BDM04**], for example, contains several places where RCN-like structures can be found. We expect that these biological architectures exploit very similar dynamics to what we investigate here.

5.1.1. Related Work. Here we do not attempt to give a thorough literature review, but whenever helpful, we will reference experimental work that points to anatomical and physiological knowledge about the brain.

Roughly 15 years ago, Amit and Brunel [**AB97**] started to look at networks like the one shown in Figure 5.1. This and subsequent work contain detailed mathematical analysis of dynamics in such networks

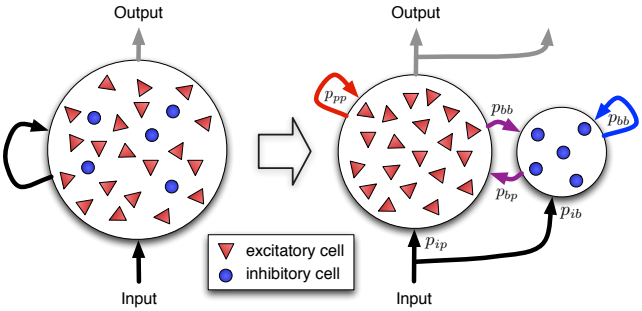


FIGURE 5.1. An RCN network. The left side shows a digram of the chaotic distribution of cell bodies in euclidian brain space. The right side shows the same network, this time with excitatory and inhibitory cells visually separated. Solid arrows symbolize a set of randomly drawn connections between cells of the interconnected or recurrent populations. The number of connections can be controlled by $\{p_{ip}, p_{ib}, p_{pp}, p_{pb}, p_{bb}, p_{bp}\}$, the set of connectivity parameters.

[**Bru00, HRB11**]. The input to their networks is assumed to be Poisson distributed, which is not necessarily the case in real world scenarios [**BMS07, DCM09, DCM11**]. In their work, each cell population consists of a monoculture of cells that are perfect copies of each other. Although this is a reasonable assumption in order to allow mathematical analysis, it is not clear how a heterogeneous network, such as the ones found in biological brains, would synchronize or desynchronize its activities. (In Chapter 7 we will have a closer look at related topics.)

Various people looked at how recurrent networks of the kind shown in Figure 5.1 can be used for various computational tasks [**RRWF10, BSF07, DGFM03**]. Although very interesting we can not deepen our discussion about these here, but we would like to single out a fairly successful approach, Liquid Computing. Liquid State Machines [**Maa07**] are maybe the most extreme attempt to use recurrent, random networks (the reservoir) for computation and learning. The reservoir and a set of trainable readout units are used to discriminate properties of the input fed to random nodes in the network. Besides working well as a model

there are even attempts to realize Liquid State Machines with biological neurons in culture [**DPH$^+$09**].

Concerning the anatomical plausibility of RCNs, studies like [**KSM05, KHP$^+$11, PBM11**] show that randomly connected, tabula rasa like networks can be found in local cortical structures. The RCNs we describe here are, in first approximation, like the networks described in these publications. Also other studies like the work by Konnerth and colleagues (see e.g. [**CLR$^+$11**] or [**VJSK11**]) suggests that neural connectivity is not targeted in an obvious way. They show that the excitatory input to cortical cells comes from many other cells that are tuned to a wide variety of features. Although it does not necessarily mean that the local connectivity is random, a random connectivity would also come with such diversity properties.

5.1.2. Anatomy of Recurrent Competitive Networks. The anatomy of the simulated networks presented here is shown in Figure 5.1(a). The connections in one such network or layer are highly recurrent, introducing positive (excitatory) as well as negative (inhibitory) coupling among the cells. The inhibitory coupling makes the excitatory cells compete, letting only the ones with more excitatory than inhibitory inputs remain active. These characteristics motivate the name "Recurrent Competitive Networks", or RCNs. We call the multi-layer RCN setup shown in Figure 7.1(b) "feed-forward RCN chain". In Chapter 7 we will use such feed-forward RCN chains in order to show how activity propagates from layer to layer.

We introduce parameters $p_{\alpha\beta}$ for $\alpha \in \{\text{i,p,b}\}$ and $\beta \in \{\text{p,b}\}$, with subscripts denoting the input population (i), the excitatory (pyramidal cell) population (p), and inhibitory (basket cell) population (b). For each potential connection between two cells in the entire network $p_{\alpha\beta}$ is the probability for the connection from α to β to exist independent of each other.

Peter's rule [**BS91**], the Canonical Microcircuit [**BDM04**], and more recent results like [**KSM03, KHP$^+$11, PBM11**] tell us to expect (locally) only little variability for the connection probability $p_{\alpha\beta}$ between cells after normalizing for their size (spatial extent).

Although it is known that the connection probability between two cells within a cortical column depends on somatic distance [**KSM03, PBM11**] and is roughly between 0.1 and 0.2, we use purely random networks, thereby neglecting these geometric aspects.

Random graphs of this kind are called Erdős-Renyi graphs and have been extensively studied for decades [**Bol01, ER60**].

Neurons in our simulations are either modeled by conductance based or by voltage based leaky integrate-and-fire (LIF) neurons [**Tuc88, DA01**]. In Chapter 6 we review the dynamics of LIF neurons in detail.

Parameters like synaptic strength and transmission delays were chosen to be biologically plausible. The justification of these parameter sets can be found in Section 6.3.

The Network Used in this Chapter. We want to fix an RCN setup for later use in this chapter. Let us, first of all, fix a network size. (In other chapters of this thesis we will use RCN setups of different sizes.)

We build a network that is large enough to be roughly comparable to its bio-anatomical counterparts. A network of 1000 excitatory and 250 inhibitory neurons is roughly as large as the neural population found in one layer in a cortical mini-column of cats or rats [**BDM04, LF07**].

The remaining, free parameters in the RCN setup are: (*i*) The connection probabilities $p_{\alpha\beta}$ for $\alpha \in \{\text{i,p,b}\}$ and $\beta \in \{\text{p,b}\}$ as mentioned in Section 5.1.2, (*ii*) the synaptic weights $w_{i,j}$ between connected pairs of cells i and j, and (*iii*) the transmission delays $d_{i,j}$ between connected pairs of cells i and j.

Although important in general, let us for now set all values $d_{i,j}$ to 0 or some small value $\epsilon \ll 1$ms. In later chapters we will reintroduce non-zero and non-uniform transmission delays. Note that there are also other neuronal properties that introduce a notion of time, like the post-spike refractory periods and synaptic rise and decay times. For details see Chapter 6. There we present in great detail the neuron model and its parameterization.

We also set all connections between a source population $\alpha \in \{i, p, b\}$ and a target population $\beta \in \{p, b\}$ to the same, fixed value $w_{\alpha\beta}$, giving us 6 such weight values for all synaptic connections in the network.

This leaves us with $6 + 6 = 12$ remaining, free parameters. Note that many pairs $p_{\alpha\beta}$ and $w_{\alpha\beta}$ are similar from a certain point of view: if a cell j receives excitatory input from twice as many cells, but with only half the effective synaptic weight, the total amount of input may likely stay the same.

The precise input characteristics do of course matter. As we will see in Chapter 8, Siegert's formula **[Sie51]** gives a detailed description of how the response of LIF neurons changes based on such changes in input distributions (see also Figure 8.1). For small changes the computed rates are similar, usually giving us some freedom in increasing (decreasing) the connection weight $w_{\alpha\beta}$ a bit and counterbalancing that by decreasing (increasing) the corresponding connection density $p_{\alpha\beta}$. Taking this into account it seems plausible that many sets of parameter can potentially lead to the the same, or at least similar, overall dynamics. Our empirical observations also support this claim.

The complete set of parameters for an RCN network with $1000 + 250$ neurons is given in Table 5.1.

TABLE 5.1. A set of parameters for an RCN setup containing 1000 excitatory and 250 inhibitory neurons. We used these parameters for all simulation results presented in this chapter. (The weight values $w_{\alpha\beta}$ are expressing the integral over the post-synaptic currents a single synapse contributes at the soma of the receiving neuron. See page 66 in Chapter 6 for a detailed explanation.)

RCN parameters							
n_p	n_b	p_{ip}	p_{ib}	p_{bp}	p_{pb}	p_{pp}	p_{bb}
		0.10	0.10	0.06	0.06	0.06	0.16
1000	250	w_{ip}	w_{ib}	w_{bp}	w_{pb}	w_{pp}	w_{bb}
		1.90	7.50	-1.80	7.50	0.90	-1.80

5.1.3. Population-Coded Input Patterns. We model a population-coded input as follows: (*i*) We fix an arbitrary ordering of the input neurons N_{inp}, giving us an additional suffix x. (*ii*) We choose a position c at which we want to center the population-coded activity profile and a spacial extent τ. (*iii*) We let all input neurons $N_{\text{inp},x}$ generate Poisson spike trains with parameters:

$$\lambda_c(x) = S \cdot e^{-|c-x|/\tau}, \tag{5.1}$$

a Laplacian function or, motivated by the visual appearance if you plot this function, double exponential.

In Chapter 7 we will change the Poisson spike event distribution of the input neurons in order to test how RCNs deal with more homogeneous or synchronized input patterns.

5.2. Results

Figure 5.2 shows how the network we have specified in Section 5.1.2 responds to a population-coded input pattern.

In the left column of Figure 5.2 we can see that the input neurons around the neuron with ID 350 produced the most input. In the second row, which shows the exact same data as the first row, but sorted by output activities, we see that these activities decay exponentially across the population. The other two columns show that the activities of the RCN cells are clustered in a similar way. Since the network connections are drawn uniformly at random, and there is no geometry or topology on which we have based the creation of individual connections, the subplots in the first row appear fairly messy.

But why does the network response look like this and how would the activity profile change by changing the fed input pattern? In this chapter we will give an intuition for this. A more detailed, mathematical analysis of the response properties of RCN networks can be found in the thesis of my colleague Christoph Krautz [**Kra12**].

5.2.1. Sparse Responses Due to Feed-Forward Inhibition (FFI).

The FFI Subnetwork. Let us first consider the subnetwork of the complete RCN shown in Figure 5.3. If we remove the self-loops from the excitatory and inhibitory subpopulations to themselves (connections of type *pp* and *bb*) as well as the connections from the excitatory subpopulation to the inhibitory subpopulation (*pb*), a feed-forward inhibition network remains. Biological evidence for the existence of FFI networks in cortex are given for example in [**PMBA$^+$09**].

Let us describe the network structure and introduce some nomenclature. Each excitatory (pyramidal) cell and inhibitory (basket) cell receives input from n_{ip} and n_{ib} input neurons, respectively. The input neurons fire with rate $\lambda_c(x)$, as described in Section 5.1.3. The synaptic weights from input neurons to pyramids and baskets are w_{ip} and w_{ib}, respectively. Each pyramidal cell is inhibited by n_{bp} interneurons. It is this source of inhibition that motivates the name feed-forward inhibition.

Before a cell creates output spikes, it is necessary to receive enough input. If synaptic weights are fixed, thresholds $\lambda_{\min_p}$ and $\lambda_{\min_b}$ exist. These values denote the minimal rate at which Poisson distributed

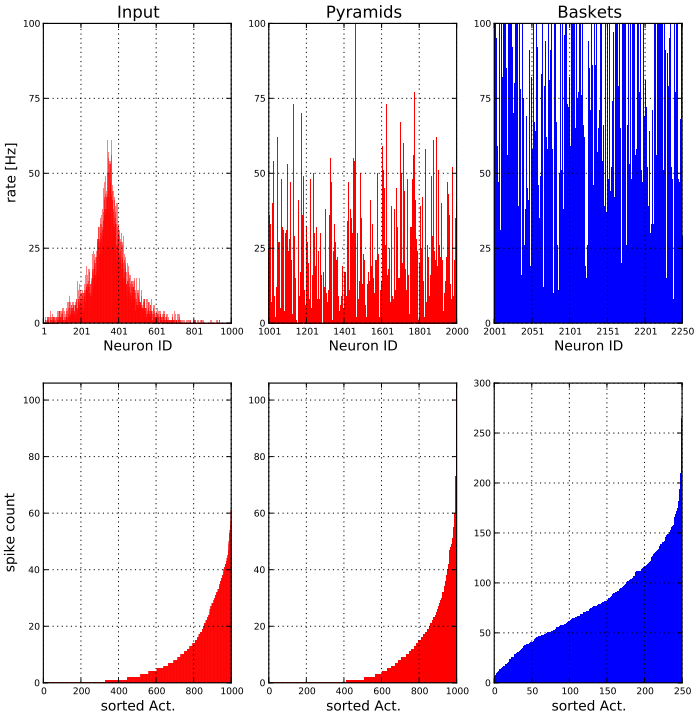


FIGURE 5.2. The characteristic response pattern of an RCN consisting of 1000 input neurons, 1000 excitatory, and 250 inhibitory cells. The precise network parameters used for the present simulation are given in Table 5.1. The left column shows the activity of the input cells, the center column the resulting activity of the 1000 excitatory neurons, and the right column the activity of the inhibitory cell population. Each figure in the upper row contains a plot showing the spike rate in Hz on the y-axis, and the neurons, in order of their creation by the simulation environment, on the x-axis. The lower row contains the same data as the upper one. The y-axis is now labeled to show total spike events recorded during the simulation time of 1.0 seconds, and the neurons along the x-axis were sorted such that the most active neuron gets displayed to the very right.

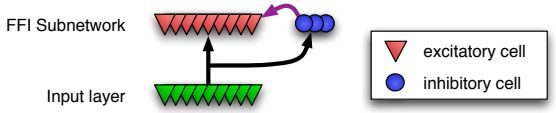


FIGURE 5.3. The FFI-subnetwork within the full RCN. See text for a discussion of the functional significance of this network.

inputs are sufficient to let the receiving pyramid or basket cell fire at all.

For the moment let us assume that this threshold is smaller for pyramidal cells than for basket cells ($\lambda_{\min_p} < \lambda_{\min_b}$). If we slowly increase the input fed into the network, the pyramids are the first to become active. When the input increases further the basket cells become active as well. This causes the pyramids, due to FFI, to lower their activities.

But the inhibitory subpopulation does not only lower the output rate of the pyramidal cells. As suggested in [**PMBA$^+$09**], basket cells that create spike events due to received feed-forward excitation will make it harder for all laterally connected cells in the entire network to become activated themselves. This is because they not only have to get an total input larger then $\lambda_{\min_p}$, but the total excitatory input has also to compensate for the inhibition they receive. (Note that this intuitive reasoning, although basically correct, draws a slightly simplified picture. The author and his colleagues are currently working on a more detailed, mathematical description of precisely these phenomena. Some of these results are contained in the thesis of Christoph Krautz [**Kra12**].)

If, in contrast to our previous assumption, basket cells are more easily excited by the external input stimulus than the pyramidal cells are ($\lambda_{\min_p} < \lambda_{\min_b}$), the story is still quite similar. Now inhibitory cells are the first to become active, making it already initially slightly harder for all other cells to become active. As long as the inhibition is not too strong, the pyramidal cells receiving most of the excitatory input will still start producing spike events. This, in turn, will engage even more inhibitory cells until no excitatory node gets excited enough to join the subset of activated pyramids. In Chapter 7 we will see an example of such a setup.

The FFI-Subnetwork in the RCN Context. Above we saw that once the pyramids and baskets are activated, every bit of additional input to the network makes the baskets more and the pyramids either more or less active, depending on the strength and excitability of the inhibitory subpopulation. This can lead to situations where (*i*) additional input makes the amount of recurrent inhibition so large that the total amount of excitatory output decreases, or to a situation where (*ii*) the input becomes so large that the refractory period of the baskets prevents a further increase of the inhibitory signal, resulting in a rapid increase of pyramidal activity. The second possibility is unlikely to play a crucial role in biological networks, sincethe natural operating range of neurons does not come anywhere close to output rates that make the refractory period be the limiting factor.

In general, an FFI network with large dynamical range [**PMBA$^+$09**] can only be built if the parameters are chosen to balance excitation and inhibition. Intuitively this means that the effective input current (total excitatory currents minus total inhibitory currents) has to be small. This might, in biological brains, be achieved by synaptic scaling, a process that is not yet fully understood but has been observed [**Tur08, GA00**].

The network dynamics are more involved if all connections shown in Figure 5.1 are present. The *pb*-connections can be seen as an additional mechanism to prevent run-away excitation – the inhibitory population is now not only sensitive to external excitation but strengthens inhibition even in response to strong internal excitation.

The *pp* connections, as we will see in detail in Chapters 9 and 10, can be used to make RCNs learn certain reoccurring input features and relations, but can also lead to a series of problems and clustering effects (that will not be explicitly shown in those chapters).

The use of the *bb* connections is less obvious. In Chapter 7 we will see that these connections are crucial for synchronization and desynchronization of population spike patterns.

Figure 5.4 shows the response of a complete RCN to a fixed input pattern of increasing input frequencies $\lambda_c(x)$. We can see that sparse, random networks of roughly the size and structure that exist in cortex, simulated using a fairly detailed LIF neuron, show very reliable output statistics that can precisely be controlled by a couple of intuitive parameters. Compared to simpler FFI-networks, the additional

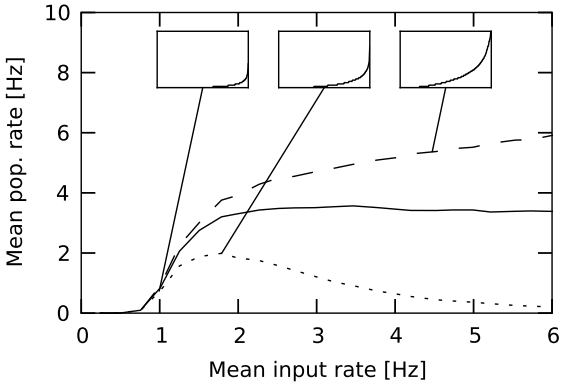


FIGURE 5.4. Response of an RCN to input patterns of increasing strength. The solid curve shows a setup where external excitation and internal (feed-forward) inhibition are almost perfectly balanced. We can see that once the input pattern is strong enough to elicit a population response, the output strength is basically independent of the input strength. Dashed and dotted curves show the response of two other RCN setups with weaker and stronger FFI, respectively. The insets show the sorted population activities as introduced in the second row in Figure 5.2.

recurrent connections in RCNs do not significantly change the observed phenomena.

The robustness of these phenomena suggest that FFI dynamics like the ones shown here could easily have been chosen by evolution to play a role in the type of computations performed in our brains.

5.2.2. Similar Input Patterns Give Rise to Similar Output Patterns. Figure 5.5 was created with the same very RCN that we used to create Figure 5.2. Only two things are different compared to Figure 5.2: (*i*) The input pattern is now centered at $c = 400$ (before it was $c = 350$), and (*ii*) the second row is not sorted by the response of this input ($\lambda_{400}(x)$), but the neuron ordering of Figure 5.2, based on the response to input $\lambda_{350}(x)$, was retained.

It is easy to see that the activity pattern, although slightly noisy, is quite similar to the one shown in Figure 5.2. Another way of visualizing

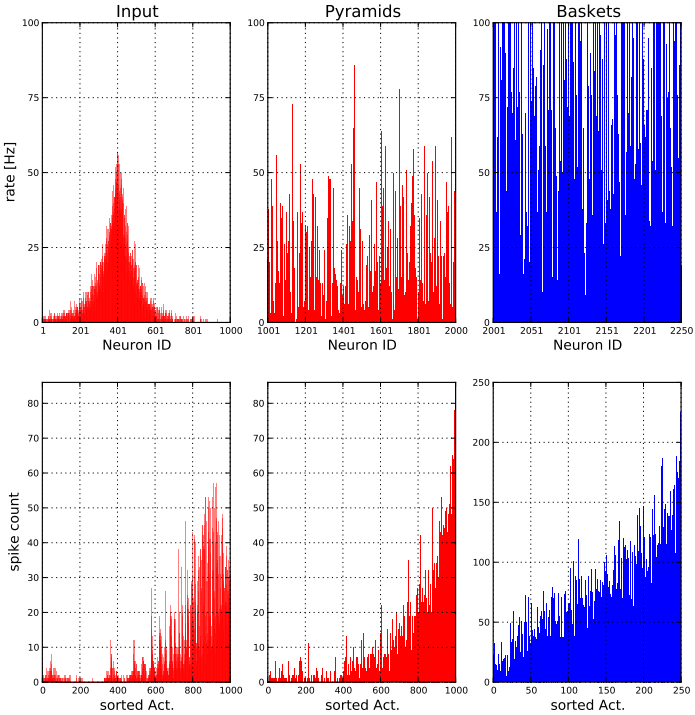


FIGURE 5.5. Same setup as for Figure 5.2. The differences are: (*i*) The input pattern is now not centered at $c = 350$, but rather at $c = 400$, and (*ii*) the second row is not sorted by the response of the given input $\lambda_{400}(x)$, but is still ordered as in Figure 5.2 according to the activities of the network in response to input $\lambda_{350}(x)$.

this effect is given in Figure 5.6. In this figure we show the input preference of the receptive fields [**HW62**] of 10 arbitrarily chosen pyramidal neurons. In order to create this figure we presented 1000 different input stimuli of the kind described in Section 5.1.3. Table 5.2 contains the set of parameters that fully describe these inputs. In total we showed

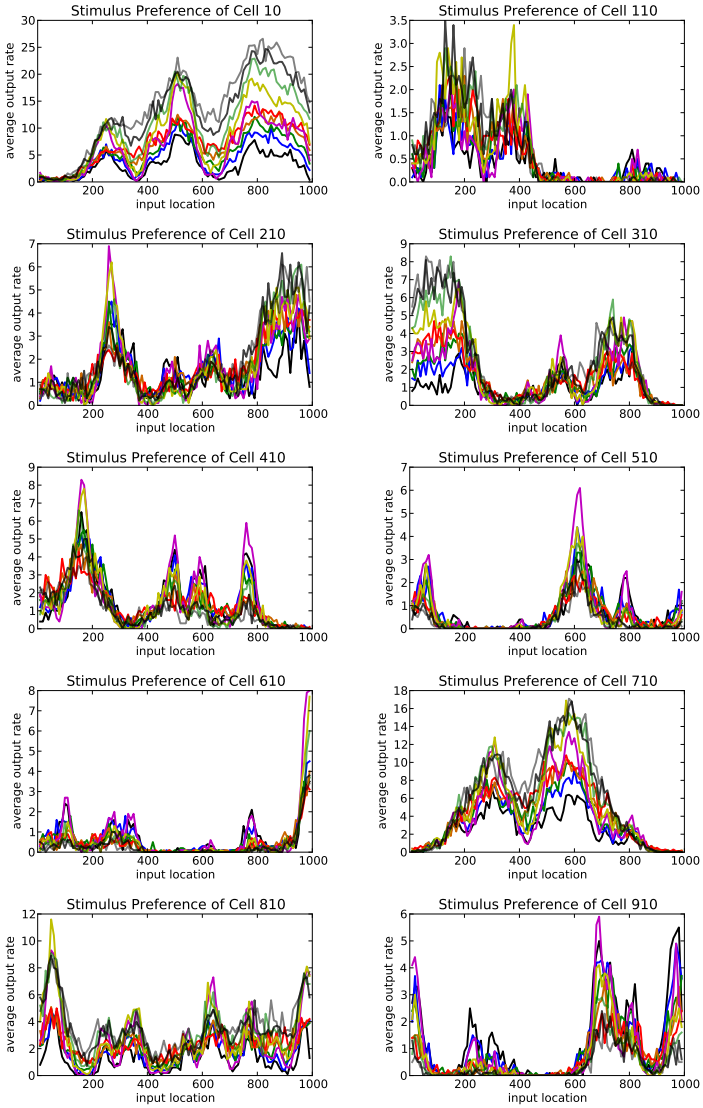


FIGURE 5.6. Tuning of 10 arbitrarily chosen pyramidal cells from the RCN used throughout this chapter. Almost every cell in the network shows strong preferences to certain input locations (x-axis), although the network's connectivity is purely random. It is interesting to see that a neuron's preferred input location(s) have some spacial extent along the x-axis.

10 different input patterns (see Table 5.2) centered at 100 different locations. Each of the subfigures in Figure 5.6 shows the tuning of one pyramidal cell in the RCN to these 100 locations.

TABLE 5.2. Parameters for all inputs used to generate Figure 5.6 and the preliminary cluster-test described in Section 5.2.3. Each combination of the given parameter values was presented to the very same RCN, giving a total of 1000 different input patterns at 100 different locations c. Note that many of these inputs are overlapping each other.

Parameter (see S. 5.1.3)	values used for cluster-test
max rate (S)	$\{30\text{ Hz}, 60\text{ Hz}\}$
input width (τ)	$\{40, 60, 80, 100, 120\}$
input position (c)	$\{10 \cdot x \mid \text{for } x \text{ from 1 to } 100\}$

We want to provide a more quantitative way of looking at the similarity of input and output patterns. For this purpose we have to define a measure of similarity between population activation patterns.

5.2.2.1. *A Similarity Measure for Input and Output Patterns.* An input pattern, as well as an output pattern, can be seen as a high dimensional vector. The spike rate of each cell in a given input or pyramidal population corresponds to one dimension in such an output vector pattern. (This is precisely the kind of data we visualized in Figures 5.2 and 5.5.)

The similarity of two vectors $\mathbf{x}$ and $\mathbf{y}$ of dimensionality d can be measured in many ways. One possibility is to compute the angle between the vectors. By standard geometry, the angle satisfies $\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}| \cdot |\mathbf{y}|}$, where:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{d} x_i y_i$$

is the dot product and $|\mathbf{x}|, |\mathbf{y}|$ are the length of $\mathbf{x}$ and $\mathbf{y}$, respectively.

Two identical, length normalized vectors give a value $(\mathbf{X} \cdot \mathbf{Y}) = 1$, two orthogonal, length normalized vectors have a dot product $(\mathbf{X} \cdot \mathbf{Y}) = 0$ (which in our case is very unlikely given that all our vectors do only have non-negative entries).

The angle gives us a very natural and easy to compute measure of similarity between two normalized population activity patterns.

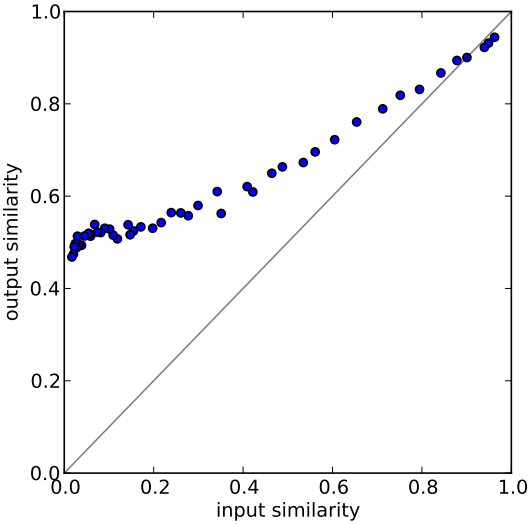


FIGURE 5.7. Visualization of a series of simulations performed with the RCN network defined in Table 5.1. Each data point represents two successive simulations where two different inputs were fed to the network. The position on the x-axis shows the similarity (angle) of the two inputs $\lambda_{c_1}(x)$ and $\lambda_{c_2}(x)$. The y-axis shows the similarity of the two output patterns elicited by these two inputs. The data shows that similar inputs give rise to similar output patterns. More dissimilar inputs lead to more dissimilar outputs. For orthogonal inputs the RCN response diverges systematically from the diagonal, meaning that the responses elicited by the two inputs are partially overlapping even when the inputs are not.

We can use this similarity measure to show how similar the response of an RCN to two stimuli is, compared to the similarity of these stimuli. (Note that this comparison works well even if the number of input neurons and pyramidal neurons do not match in number.) Figure 5.7 contains similarity tests performed on a series of simulations. (Again we have used the RCN network defined in Table 5.1.) Each data point represents two successive simulations where two different inputs were fed to the network. The position on the x-axis shows the similarity of

the two inputs $\lambda_{c_1}(x)$ and $\lambda_{c_2}(x)$. The y-axis shows the similarity of the two output patterns elicited by these inputs.

For data points on the diagonal, the similarity of the inputs matches the similarity of the outputs. We can see that very similar input patterns lead to very similar outputs. If the two inputs become more distinct, the output patterns start to systematically diverge from the diagonal: the output patterns become more similar than the input patterns. (Note that the similarity of two random vectors is also not 0.)

The major reason for the values in Figure 5.7 to be placed above the diagonal can be explained by the distance measure we have used: the dot product ($\mathbf{X} \cdot \mathbf{Y}$). The dot product equals 0 if and only if the vectors ($\mathbf{X}$ and $\mathbf{Y}$) are orthogonal, i.e. only if these vectors do not share non-zero dimensions. (Note that this is only true because our activity vectors cannot contain negative values.) In Figure 5.2 we see that our network's activity vector has nonzero entries for more then half its dimensions already. Hence we cannot find two such vectors with $(\mathbf{X} \cdot \mathbf{Y}) = 0$.

Besides that, some of the pyramidal cells do, by chance, have more excitatory connections (or maybe also less inhibitory inputs from the basket cells) and are therefore more active on average than all the others. This uneven activation of the neurons in an RCN gives rise to the higher degree of similarity in resulting output patterns.

Note that all this does not necessarily mean that similar input patterns cannot be distinguished. Two vectors with a dot product well above zero can still be quite different and easy to discriminate at a subsequent processing stage.

5.2.3. Different Inputs Remain Distinguishable. Since the connectivity in RCN networks is stochastic, one might be concerned that different input patterns could lead to similar output activities. If this would be the case, we might lose information and could at later processing stages not distinguish similar but different inputs any more.

Although we see in Figures 5.5 and 5.6 that similar inputs have the tendency to excite the network in similar ways we cannot be satisfied with the relatively informal, visual clues we get from these figures. The question is how well inputs could be discriminated if all you have is the activities of the RCN's pyramidal population.

We think that this boils down to the question of how well tabula rasa [**KSM05, KHP$^+$11, PBM11**] type networks can discriminate inputs.

If the "tabula rasa" hypothesis in **[KSM05]** holds it seems highly desirable that the initial sensory networks should not lose information about the inputs they receive. If a quantitative analysis of RCNs would show that they can indeed distinguish even fairly similar input patterns, then the tabula rasa hypothesis could at least in principle be correct.

Although we are working on such a rigorous quantitative analysis we are not yet able to present clear-cut results. Instead we will report on preliminary tests we performed on the dataset created for Figure 5.6.

In order to demonstrate the robustness and reliability of RCNs, we created a set of 1000 input patterns (see Table 5.2). We fed each of these 1000 patterns in 10 consecutive runs to the RCN defined in Table 5.1. Note that each of these 10 individual simulations per input pattern are not identical. This is because input patterns are only defined in terms of Poisson rates rather than precise spike timings and spike counts.

For each input we collected the elicited outputs into a vector of length 1000. We then used a k-means classifier to see whether all 1000 ten-point clusters were stable under the k-means algorithm, or whether these clusters would turn out to intersect in the 1000-dimensional vector-space. In order to do so we initialized the 1000 clusters by labels representing the 1000 different input patterns.

The result of this clustering test was clear-cut. We could classify 100% of the input patterns correctly. This means that the 1000 clouds containing 10 points each were embedded in the RCN-output-space in such a way that each point was closest to the center of its own cluster. In other words: each input pattern was causing a distinct output pattern without giving rise to any confusion between them.

5.2.4. A Superposition of Inputs Is a New Input Pattern. What might happen if we not only show one of the inputs described in the last section but instead combine two of them? Since we know about the FFI-aspects of RCNs we know how the overall network activity will change (Figure 5.4), but how would this activity be distributed?

Figure 5.8 shows the responses of an RCN (i) to an input stimulus A, (ii) to an input stimulus B, and (iii) to the superposition (sum of individual Poisson rates) of the stimuli A and B, stimulus AB.

We see that the responses to all three input patterns are quite different. The pairwise similarity (dot-product) of the normalized output patterns is 0.25 for patterns A and B, 0.70 for patterns A and AB, and 0.58

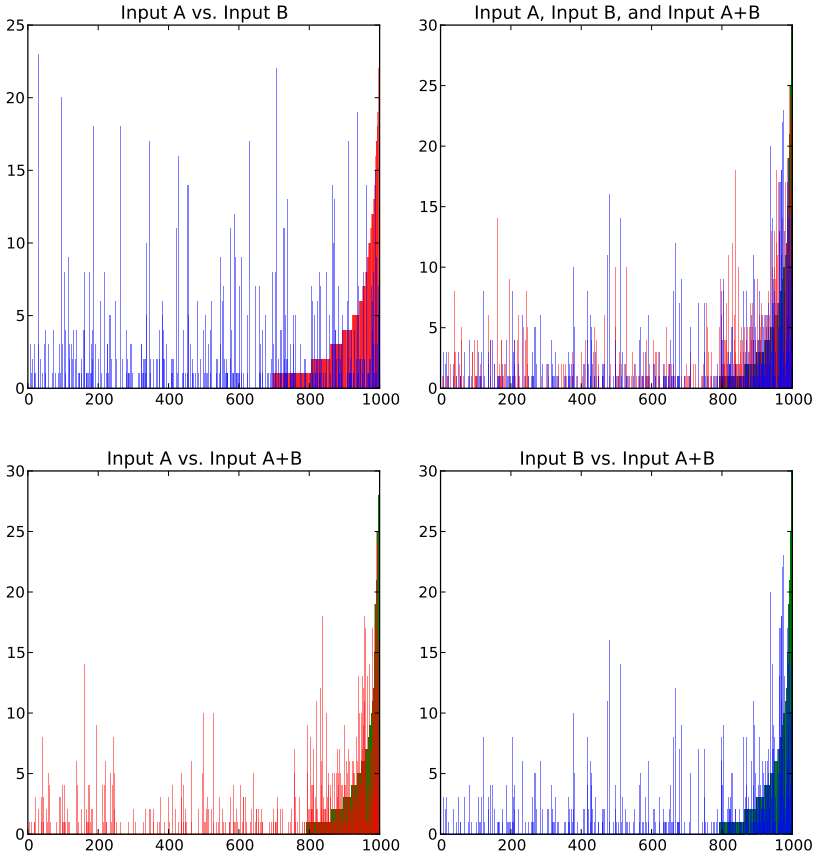


FIGURE 5.8. Response of an RCN (*i*) to an input stimulus A, (*ii*) to an input stimulus B, and (*iii*) to the superposition of the stimuli A and B, stimulus AB. For all subfigures, the pyramidal cells along the x-axis are sorted according to the activities elicited by the last-mentioned input in the title. The output pattern corresponding to input A is shown in red, the one for input B in blue. Stimulus AB is shown in green. We can see that the responses to all three input patterns are quite different. The pairwise similarity (dot-product) of the normalized output patterns is 0.25 for patterns A and B, 0.70 for patterns A and AB, and 0.58 for patterns B and AB.)

for patterns B and AB (see Section 5.2.2 for an explanation of this similarity measure), showing that the response to a superposition of two inputs gives rise to an output that shares a certain similarity with both of them but is still very well distinguishable from both of them.

5.3. Discussion

In this chapter we introduced Recurrent Competitive Networks (RCNs). We have seen that these networks are inspired by our current anatomical knowledge [**PBM11, KHP+11, PMBA+09, CLR+11, VJSK11, ASLB07, KSM05, BDM04, BS91**] and that it is likely to find RCNs in different places in the CNS of various species.

We then fixed a representative instance, for which we have provided a complete set of parameters. We used this model throughout the chapter to demonstrate typical RCN dynamics.

Comparing these dynamics with those of classical competition models (like the ones we used in Part 1 of this thesis), we can see both, similarities and differences.

5.3.1. Similarities Between RCNs and CCMs. In Section 5.2.1 we showed that the output of an RCN is sparse, meaning that most cells show below-average activity levels. This, in fact, is similar for population-coded cell assemblies, first described by Georgopoulos [**GKCM82**]. For further details on population codes see [**DLP99, PDDL99, DLP01**].

Each cell in a population-coded cell assembly has a preferred input stimulus. For stimuli that are similar to the preferred stimulus, the cell responds quite actively. This characterization is nicely modeled by CCMs like divisive inhibition or convolution based models (see Part 1).

Cells in RCNs also have the property that they are activated by similar input patterns (see Section 5.2.2). Later, in Chapter 9, we will show that RCNs can be modified by a simple learning rule so that they learn the topology of the input space, making them even more similar to the CCMs of Part 1[1].

Another similarity of CCMs and RCNs is that the output strength of many CCMs, like divisive inhibition or convolution based methods,

[1] This, as we will see in Chapters 9 and 10, is true even though the topology had to be hard-wired into the CCMs.

scales sub-linearly with the input strength (this follows e.g. from the model descriptions given in [**DLP01**] and [**Ama77**]).

Not only do the dynamics of RCNs match fairly well with biophysical results like the ones in [**PDDL99**], by design also the network structure of RCNs reasonably matches the known anatomy of local cortical circuits [**KSM03, PBM11, BDM04**].

5.3.2. Differences between RCNs and CCMs. The biggest difference between CCMs and RCNs is in their response to a superposition of inputs. There is not yet a scientific consensus for how biological cell populations respond to the superposition of two inputs-

Classical competition models come with diverse responses to input patterns like the ones we used to create Figure 5.8. To our knowledge there is no alternative model described in the literature that shows response patterns qualitatively similar to RCNs. (But mind the plethora of related work we cited in the introduction of this chapter. Most of this work had a different scientific focus, but we expect that basically all of them show comparable dynamics to what we have described here.)

Nevertheless, we think that the RCN type of network response to a superposition of individual input patterns is remarkably similar to experimental data obtained in the Institute of Neuroinformatics (data not published when this thesis was finalized).

It is very interesting to see how recurrent networks that may be quite similar to real brain networks respond to external stimulation. Still, the question of how such modules could collaborate and interact in our heads in order to let us think and reason remains a challenging, largely unanswered scientific quest.

In the following chapters we elaborate on possible answers to this question. We discuss possible extensions and couplings of RCNs to build systems that show richer dynamics, being capable of performing learning, inference, cue-integration, and decision tasks.

Before building larger networks, though, we will review biologically plausible parameter regimes for leaky integrate-and-fire neurons.

CHAPTER 6

Biologically Plausible Integrate and Fire Parameters

6.1. Introduction

When building a neural network, especially when attempting to model biological networks, one needs to choose one of many possible neuron models. Within the last decades many such models have been used, ranging from simplistic units such as McCulloch-Pitts neurons [**WL90, Ros58, MP43**] up to complex, multi-compartmental models as described and used in [**YKA89, EWL$^+$91, HC97, BB03**].

While simple models do not capture the dynamics of real neurons very well, complex neurons can be so computationally expensive that efficient simulation of large neural networks is simply not feasible. Which neuron model to pick depends very much on the modeling task at hand [**Izh04**].

Since we are interested in biologically plausible, recurrent networks like the recurrent competition networks (RCNs) introduced in the previous chapter, we want to pick a neuron model that describes biological neurons as well as possible, while allowing us to efficiently simulate networks that consist of several hundred or even thousands of neurons. A natural choice is the so-called leaky integrate-and-fire neuron (LIF-neuron) [**RLCL$^+$03, Tuc88, DA01**].

But the need for design decisions does not end here. In order to instantiate and later also simulate our network of choice we have to decide which simulation environment to use or if we want to go and build one ourselves. Since there are many such tools available [**EHM08, BB03, DG02, WJW$^+$02, HC97**] it is most likely a good idea to pick one of those.

In our case we chose the *Neural Simulation Tool, NEST* [**EHM08, DG02**] for most of our simulations. NEST already contains a LIF-neuron model, but we extended their implementation significantly in order to include additional parameters and synaptic dynamics.

A very nice and quite detailed review of existing neural simulation strategies and tools can be found in [**BRC$^+$07**]. Rudolph and Destexhe elaborate in [**RD06**] on whether or not neural simulations can mimic the dynamics of real neural networks and what has to be considered in order for them to do so. Izhikevich offers a nice overview of the most important spiking neuron models and lists their main features and limitations in [**Izh04**].

6.2. The Neuron Model

All model neurons used in the simulations performed here are leaky integrate-and-fire (LIF) neurons [**Tuc88, DA01**]. It is known that LIF neurons can match cortical cell physiology quite well [**RLCL$^+$03**]. We have implemented them to be compatible with the NEST framework [**EHM08, DG02**].

In the following sections we will briefly describe the basic concepts of LIF neurons. A detailed, well written introduction to LIF neurons, and other neuron models, can be found in the book by Gerstner and Kistler [**GK02**].

6.2.1. Leaky Integrate and Fire Dynamics. A LIF neuron is modeled by a fluctuating *membrane potential* V_m that decays exponentially, with *decay rate* τ_m, to its *resting potential* V_rest. Synaptic inputs can raise or lower V_m. (How synaptic inputs precisely influence the membrane potential can be found in Sections 6.2.3 and 6.2.4.) If V_m rises above the *threshold potential* V_th, the cell elicits an action potential (spike) and sends this event to all neurons along its axon. The *transmission delay*, i.e. the time it takes to transmit this event to the soma of the receiving cell, can be set individually for each receiving cell. After generating an action potential, a neuron enters the so called *absolute refractory period* – a phase that lasts $\mathrm{t_{ref}}$ milliseconds. During this time the LIF neuron is not integrating synaptic input and is therefore guaranteed not to generate further action potentials. After the absolute refractory period the cell's membrane potential is reset to V_reset.

The dynamics of a standard leaky LIF model can be described by the following differential equation:

$$\tau_{\mathrm{m}} \frac{dV_{\mathrm{m}}(t)}{dt} = -(V_{\mathrm{m}}(t) - V_{\mathrm{rest}}) + R_m I(V_m, t) \ , \tag{6.1}$$

where R_m is the membrane resistance and $I(V_{\mathrm{m}}, t)$ the received synaptic current at time t. Note that the synaptic current can depend on V_{m}, the current membrane potential.

6.2.2. Synaptic Integration and Transmission Delays. Every connection $c_{i,j}$ from neuron i to neuron j has a transmission delay $\delta_{\mathrm{i,j}}$ assigned to it. Precisely $\delta_{\mathrm{i,j}}$ milliseconds after neuron i generated an action potential, the synaptic input $I(t)$ starts arriving at the soma of cell j. $I(t)$ can be model in different ways. The two most common models simulate the current flow directly by specifying the current or indirectly by changing the conductance of a cell j's membrane (see e.g. **[GK02]** for a detailed explanation). These two models can lead to different network dynamics [**DRFS01, MBG04**]. The conductance based model often provides a better fit to biological neural dynamics.

Let us have a closer look at current and conductance based synaptic integration.

6.2.3. Current Based Synaptic Input Integration. If synaptic input is modeled on the basis of current given by the function $I(t)$, the total change of the postsynaptic membrane potential V_{m} depends only on I and τ_{m}. This means that the influence of one synapse on V_{m} is always the same, independent of the current value of V_{m}.

The synaptic currents in biological neurons depend on the types of receptors that are activated by the released types of neurotransmitters. Each receptor evokes synaptic currents of different amplitudes and durations and therefore has to be modeled by a separate term in Equation 6.1.

We explicitly model two excitatory receptor types, AMPA receptors and NMDA receptors, and one inhibitory receptor, the $\mathrm{GABA_A}$ receptor. The next paragraphs will give all necessary details about the precise dynamics of these three receptor types.

6.2.3.1. *Excitatory Post-Synaptic Currents (EPSCs).* The EPSC is the sum of two separate currents I_{AMPA} and I_{NMDA}, corresponding to the currents from the glutamatergic receptors α-Amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid (AMPA) and N-methyl-D-aspartate (NMDA).

We model $I_{\mathrm{AMPA}}(t)$ currents by a single exponential:

$$I_{\mathrm{AMPA}}(t) = \begin{cases} c_{\mathrm{AMPA}} e^{-(t-t_0)/\tau_{\mathrm{AMPA}}} & , \ t \geq t_0 \\ 0 & , \text{otherwise} \end{cases}, \tag{6.2}$$

as for example described in **[Wan99, RBS+67]**. The factor $c_{\mathrm{AMPA}} = \frac{\alpha_{\mathrm{AMPA}}}{R\,\tau_{\mathrm{AMPA}}}$ is chosen such that the total evoked voltage is α_{AMPA}, the parameter for the synaptic efficacy with respect to AMPA.

The NMDA contribution, in contrast, is modeled by a double exponential[1] with parameters $\tau_{\mathrm{NMDA,rise}}$ and τ_{NMDA} for rise and decay time constants respectively. (See again **[Wan99, WvRM+00]** for further details.) The area under the function $I_{\mathrm{NMDA}}(t)$ is similarly normalized to be α_{NMDA}, the parameter for the synaptic efficacy with respect to NMDA.

In total, a single presynaptic spike induces a total potential change of $\alpha_{\mathrm{ex}} = \alpha_{\mathrm{AMPA}} + \alpha_{\mathrm{NMDA}}$, α_{ex}usually being called the synaptic weight or total synaptic efficacy. Synaptic currents of multiple synaptic spikes are summed up linearly.

6.2.3.2. *Mixing of AMPA and NMDA Currents.* We used data of the compound postsynaptic currents (PSC) of excitatory cells **[MLF+97, BGC03, TWWB02]** to deduce the relative contributions of AMPA and NMDA currents at single synapses. We did so by finding the best fit to measurements published in these papers.

In Section 6.3 (Table 6.1) we show the results of our data-fitting analysis.

6.2.3.3. *Inhibitory Post-Synaptic Currents (IPSCs).* IPSCs are modeled by a GABA current. This current is, like the AMPA current discussed before, modeled by a single exponential with one parameter (τ_{GABA}) and a total synaptic efficacy of α_{in} **[Wan99, GH98, WGTR+02, BCHS95]**.

In biological systems GABA currents are further subdivided into GABA_A and GABA_B currents, which operate like AMPA and NMDA on slightly different timescales. For simplicity we do not model these two currents explicitly.

[1] Double exponential, in this context, denotes a fast exponential rise time and a slower exponential decay. Mathematically this can be expressed by $f(t) = c_{\mathrm{NMDA}} \cdot (1 - e^{-(t-t_0)/\tau_{\mathrm{NMDA,rise}}}) \cdot e^{-(t-t_0)/\tau_{\mathrm{NMDA}}}$.

6.2.4. Conductance Based Synaptic Input Integration. The assumption that the influence of a synapse on the membrane potential V_{m} of the postsynaptic cell is always the same, independent of the actual value of V_{m}, is not true for real neurons. Since synapses do not actively transmit currents, but rather influence the permeability of the postsynaptic cell to certain positively or negatively charged ions, the effective current flow depends on the concentration difference of these ions in the intracellular and extracellular fluid (see e.g. **[GK02]** for a detailed explanation).

The model can account for this if the function describing the synaptic currents that flow into the postsynaptic cell depends not only on t but also on V_{m}, as seen in Equation 6.1. In such a model the synaptic current I is given by

$$\tau_{\mathrm{m}} I(V_m, t) = -G_{\mathrm{ex}}(t)(V_{\mathrm{m}}(t) - E_{\mathrm{ex}}) - G_{\mathrm{in}}(t)(V_{\mathrm{m}}(t) - E_{\mathrm{in}}) \,, \qquad (6.3)$$

where E_{ex} and E_{in} are the reversal potentials of positively charged (sodium) ions and negatively charged (potassium) ions, respectively.

An excitatory synaptic event contributes a relative conductance $G_{\mathrm{ex}}(t)$, modeled by a single exponential for AMPA and by a double exponential for NMDA. The parameters τ_{AMPA}, τ_{NMDA}, and $\tau_{\mathrm{NMDA,rise}}$ are the same as for the current-based synapse model (see Section 6.2.3). We normalize the area under $G_{\mathrm{ex}}(t)$ and $G_{\mathrm{in}}(t)$ to $\hat{g}_{\mathrm{AMPA}}$ and $\hat{g}_{\mathrm{NMDA}}$, respectively. In this way $\hat{g}_{\mathrm{AMPA}}$ and $\hat{g}_{\mathrm{NMDA}}$ become the parameters for the relative contribution of AMPA mediated versus NMDA mediated excitatory synaptic input. The total excitatory synaptic efficacy is $\hat{g}_{\mathrm{ex}} = \hat{g}_{\mathrm{AMPA}} + \hat{g}_{\mathrm{NMDA}}$, the sum of these two values.

An inhibitory synaptic event contributes a relative conductance $G_{\mathrm{in}}(t)$ modeled by a single exponential transmitted by GABA. The parameter for the exponential, τ_{GABA}, is the same as for the current model (Section 6.2.3). The area under the function $G_{\mathrm{in}}(t)$ is normalized to be $\hat{g}_{\mathrm{in}}$, the parameter for the synaptic efficacy with respect to inhibitory NMDA currents. The total inhibitory conductance is given by the sum of all GABA contributions of all active synaptic events.

6.3. Bio-plausible Parameters

Biophysical experiments usually show huge variances on many measurable properties of single neurons and neural populations. In this section we review the literature, finding many examples of this variance. Of

course we cannot cover the vast amount of existing biophysical and anatomical data, so we choose a representative sample.

6.3.1. Neuronal Properties. Setting up a leaky integrate-and-fire (LIF) neuron to behave like a real cortical cell is difficult. Besides the fact that LIF neurons can only model regular spiking cells [**Izh04**], there is currently no consensus on what parameters should be used when one wants to model cell type X of brain area Y, for any X or Y.

We provide an overview of experimental data on cortical pyramidal cells and cortical basket cells that gives indications for how to set the LIF neuron's parameters. All the collected animal data was measured in cats, ferrets, and rodents.

6.3.1.1. *The Resting Potential.* There is a long list of publications containing in vitro resting potential data in various animals [**Sch78, CGP82, MCLP85, DP89, TW93, BCHS95, SS95, BSHS96, FVM99, GH01, WGTR$^+$02, BGC03, GWGR03, GBKU$^+$04, GBKU$^+$04, RMAH07, KHOC07, KO08, LL10**]. The in vitro spectrum of measured resting potentials in pyramidal cells ranges from -84 ± 7 mV to -61.0 ± 5.4 mV, and for inhibitory cells from -73 ± 2 mV up to -64.0 ± 5.5 mV.

In vivo experiments are less frequent. For inhibitory cells they tend to show slightly lower resting potentials. This offset to the in vitro values can possibly be explained by tonic background noise present in vivo (but not in vitro) [**LRDL06**].

In vivo data from cat is, for example, available in Baranyi et al. [**BSW93**]. Margrie et al. [**MBS02**] provide in vivo data in mouse and rat somatosensory cortex, and Degenetais et al. [**DTGG02**] in rat prefrontal cortex. [**DTGG02**] also classify the cells they are recording from on the basis of their spiking properties.

The in vivo spectrum of measured resting potentials in pyramidal cells is remarkably high, ranging from -83.8 ± 5.2 mV to -64.2 ± 3.9 mV. Note that not only do different studies show very different parameter ranges, but each individual study comes already with remarkably high standard deviations.

For inhibitory cells Fricker et al. [**FVM99**] give a detailed analysis. They measure an in vitro resting potential of -74 ± 9 mV, but argue that the technique they use might yield values about 13 ± 6 mV more negative compared with whole cell recordings of the same neurons. This

would, as before, hint at a lower resting potential for inhibitory cells in vivo.

In spiking neuronal network models values close to −65 mV are most frequently used [**BDMK91, LRDL06, Wan99, BMS07**] for modeled pyramidal neurons. Inhibitory (basket) cells are usually set to similar values. Lansner and colleagues (in [**LRDL06**] as well as in preceding publications) distinguish between resting potentials with tonic background noise and in its absence (−75 mV vs. −60 mV).

6.3.1.2. *The Threshold Potential.* In vitro data for the threshold potential of individual excitatory or inhibitory neurons can, for example, be found in [**SS95, FVM99, FM00, RMAH07, KO08, LL10**].

Detailed in vivo data in rat prefrontal cortex can be found in Degenetais et al. [**DTGG02**]. For pyramidal cells they find parameter ranges from -50.8 ± 5.8 mV to -48.5 ± 3.9 mV.

In vitro data for basket cells can be found in [**FVM99**] (-49 ± 8 mV) or [**FM00**] (-38 ± 6 mV).

For pyramidal cells in spiking neuronal network models values around −52 mV are often used [**LRDL06, HT05, Wan99**]. For basket cells values used in models can also be around −52 mV (e.g. [**Wan99**]).

6.3.1.3. *Sub-Threshold Range.* In order for a model neuron to follow the dynamics of its biological counterpart the absolute values of resting and threshold potential are less influential than their relative distance to each other. The distance $A = |V_{\text{rest}} - V_{\text{th}}|$ indicates how much input has to be integrated by a neuron in order to generate a spike.

The high variability in all mentioned measurements gives us a fairly broad range of potentially plausible values for A.

For pyramidal neurons the reviewed literature allows values in the large range of $A \in [10, 35]$ mV. For basket cells this range is slightly different but even larger: $A \in [11, 44]$ mV.

Most modeling work we found is using setups within the plausible ranges for A. Heinzle et al. [**HH07**], just to mention one example, use a value of $A = 20$ mV for pyramidal as well as for basket cells.

6.3.1.4. *Membrane Time Constant (Leakage).* A vast amount of data can be found for membrane time constants in various places in different animals and cell types. For excitatory cells we reviewed [**Sch78, CGP82, MCLP85, TW93, BCHS95, BGC03, GBKU$^+$04,**

PGBZ$^+$06, KHOC07, KO08, LL10], for inhibitory cells [**MCLP85, TDW93b, BCHS95, BSHS96, WGTR$^+$02, PHT03, BGC03, GBKU$^+$04, GBKP$^+$05, PGBZ$^+$06, KHOC07**]

For pyramidal cells the cited studies offer values ranging from $6.83 \pm 1.59ms$ [**TW93**] to $30 \pm 6ms$ [**KHOC07**]. A majority of this data supports a membrane time constant of about 17.4 ± 4.5 to 22.3 ± 3.6 ms: [**PGBZ$^+$06, GBKU$^+$04**] in monkey, [**MCLP85, Sch78**] in guinea pig, and [**PGBZ$^+$06, BGC03**] in rats.

For inhibitory cells values range from $4.2 \pm 2.1ms$ [**TDW93b**] to $20.41 \pm 8.25ms$ [**WGTR$^+$02**].

In spiking neuronal network models like [**HH07**] or [**Wan99**] values of roughly 20 ms for pyramidal cells and 10 ms for inhibitory cells are used.

6.3.1.5. *Refractory Period.* The reviewed literature did not contain definitive statements about refractory times. Many LIF-models use 2 ms for pyramidal cells and 1 ms for basket cells.

6.3.2. Synaptic Properties. We not only want the parameters of our model neuron to match physiological data, but also the synaptic dynamics should be modeled as well as possible. In this section we will continue our literature review focusing on synaptic properties.

The "strength" of a synaptic connection, often called its "synaptic efficacy" depends on multiple factors. Cortical excitatory synapses use mostly glutamate as the neurotransmitter. On the postsynaptic site, glutamate reacts with two receptor types, (*i*) AMPARs (α-amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid receptors) and (*ii*) NMDARs (N-methyl-D-aspartate receptors). These receptors, once activated, locally change the permeability of the membrane such that positively and/or negatively charged ions can pass. It is this ion flow that gives rise to the post-synaptic currents (PSCs) we will describe below slightly more detail.

Inhibitory synapses, using GABA or Glycine as their neurotransmitter, function in a very similar way. The relevant receptors are GABA_A receptors and GABA_B receptors and these will also be discussed in more detail below.

In order for a synapse in our simulations to have a well-defined synaptic efficacy, we use the total synaptic current that flows into the postsynapic soma after this synapse receives an action potential. We denote this

value as the synaptic efficacy w. Dependent on the nature of the synaptic currents, w will be denoted with appropriate suffixes like w_{AMPA}, w_{NMDA},w_{GABA}, or w_{ex} and w_{in} for total excitatory and inhibitory currents.

6.3.2.1. *AMPA mediated PSPs and PSCs.* AMPA mediated currents are fast currents. They are usually modeled by a instantaneous rise to a maximum initial current followed by an exponential decay controlled by the parameter τ_{AMPA}.

In [**KHOC07, WvRM$^+$00, HNPS90a, HSN90**] we found measurements of AMPA mediated excitatory postsynaptic currents (EPSC) in rats, and [**BGC03**] measure directly the resulting change in excitatory postsynaptic potential (EPSP).

Reported EPSC amplitudes for pyramidal cells range from 25.5 ± 2.0 pA [**WvRM$^+$00**] to 40.7 ± 5.5 pA [**KHOC07**], for interneurons (inhibitory neurons) 75.2 ± 10.2 pA [**KHOC07**]. Values for τ_{AMPA} range from 3.1 ± 0.8 ms [**HNPS90a**] to 7.11 ± 0.93 ms [**KHOC07**] for pyramidal neurons, and 2.68 ± 0.27 ms [**KHOC07**] for interneurons.

The EPSP properties reported in [**BGC03**] (rat, barrel cortex) show 1.1 ± 1.1 mV amplitudes (ranging from 0.2 to 4.1 mV) with a 0.88 ± 0.26 ms $20 - 80\%$ rise time and a 12.3 ± 2.2 ms half-width for pyramidal neurons. For fast spiking interneurons they report EPSP amplitudes of 2.2 ± 2.2 mV (ranging from 0.2 to 10.1 mV) with a 0.37 ± 0.11 ms $20 - 80\%$ rise time and a 4.9 ± 1.9 ms half-width. For cat (visual cortex, Layer 4), [**THMSJ99**] report, depending on the type of the targeted excitatory neuron class and the origin of the input, EPSP amplitudes between 0.214 ± 0.138 mV and 1.018 ± 0.768 mV, with rise times of 1.1 ± 0.3 ms and 1.2 ± 0.4 ms, and 12.3 ± 5.0 ms and 11.4 ± 4.3 ms half-widths respectively.

This data is interesting because it can be used to cross validate the combined dynamics of our LIF-neuron with the used synaptic parameters.

A model by Wang et al. [**Wan99**] uses AMPA currents with a rise time of roughly 0.2 ms, and a decay time $\tau_{\mathrm{AMPA}} = 2$ ms.

6.3.2.2. *NMDA mediated PSCs.* NMDA mediated currents are slow currents. They are usually modeled by a double exponential[2] with parameters $\tau_{\text{NMDA,rise}}$ (parametrizing the current rise time) and τ_{NMDA} (parametrizing the current decay).

In [**WvRM$^+$00, KY96, HNPS90a, HSN90**] we found measurements of NMDA mediated EPSCs in mouse and rat.

Reported EPSC amplitudes for pyramidal cells range from 6.0 ± 0.6 pA [**WvRM$^+$00**] to 45.9 ± 17.3 pA [**KY96**]. In general we observed a tendency for higher values in older animals. Values for $\tau_{\text{NMDA,rise}}$ should be chosen to match rise times between 11.8 ± 4.9 ms [**HNPS90a**] and 15.5 ± 2.5 ms [**WvRM$^+$00**]. Decay rates τ_{NMDA} are reported in the range between 93 ± 38 ms [**HNPS90a**] and 163.7 ± 50.9 ms [**WvRM$^+$00**]. Hestrin et al. report in [**HSN90**] that the decay phase of NMDA currents might contain two separate exponential components, a fast component with a decay rate around 23.5 ± 3.8 ms, and a slow component with a decay rate of about 123 ± 38 ms.

A model by Wang et al. [**Wan99**] uses NMDA currents with a rise time of roughly 8 ms, and a decay time constant τ_{NMDA} of 80 ms.

6.3.2.3. *AMPA / NMDA Ratio.* The postsynaptic process contains both receptor types: AMPARs and NMDARs. The precise EPSP's time course depends on the absolute and relative quantities of the present receptors. Here we are mainly interested in finding biologically plausible values for the ratio of AMPA and NMDA ($\rho_{\text{AMPA}} = \frac{w_{\text{AMPA}}}{w_{\text{AMPA}}+w_{\text{NMDA}}}$) receptors in the postsynaptic membrane.

We found data about this ratio for synapses onto pyramidal neurons in [**WvRM$^+$00, MLF$^+$97, MSTN03**], and in a review by Thomson et al. [**TD94**] for synapses onto inhibitory neurons. For pyramidal neurons ρ_{AMPA} ranges from $69 \pm 7\%$ [**MLF$^+$97**] to 81.3% [**WvRM$^+$00**]. For excitatory connections onto inhibitory neurons the NMDA component seems to be much smaller or even absent [**TD94**].

6.3.2.4. *Mixed AMPA / NMDA PSPs and PSCs.* Mixed EPSCs are given in [**GH01, GH98, HNPS90b, SS95, MSTN03**], while data about mixed EPSPs can be found in [**TDW93a, TW93, TWWB02, PGBZ$^+$06, HHSZ03, GH01, FM00, MLF$^+$97, STHM$^+$96**]. The data given in these publications is important to validate the combined

[2] Double exponential, in this context, denotes a fast exponential rise time and a slower exponential decay. Mathematically this can be expressed by $f(t) = c_{\text{NMDA}} \cdot \left(1 - e^{-(t-t_0)/\tau_{\text{NMDA,rise}}}\right) \cdot e^{-(t-t_0)/\tau_{\text{NMDA}}}$.

dynamics of our modeled AMPA and NMDA currents and the parameters used for describing die LIF neuron itself.

In [**THMJS98**] we found layer 4 in-vitro data obtained in slices of cat visual cortex regarding mixed EPSPs from local (layer 4) excitatory cells on to basket cells. At -60 mv the EPSP amplitude was 1.471 ± 1.321 mv at a half-width of only 3.0 ± 1.1 ms.

The model by Hill and Tononi [**HT05**] uses AMPA amplitudes of 0.1 pA and AMPA rise and decay rates of 0.5 ms and 2.4 ms, respectively. For NMDA currents they use amplitude values of 0.075 pA and rise and decay rates of 4 ms and 40 ms.

6.3.2.5. *GABA mediated PSPs and PSCs.* As mentioned before, there are fast and slow GABA currents. Although we do review both types of currents, our model will only contain fast GABA currents, since slow GABA_B currents are slowing down simulations significantly without influencing the type of network interactions we are interested in. Fast GABA mediated currents are usually modeled like AMPA currents, by an instantaneous rise to a maximum initial current and an exponential decay controlled by the parameter τ_{GABA}.

In [**GH98, BCHS95, WGTR$^+$02**] we found measurements of GABA mediated inhibitory postsynaptic currents (IPSC) in rats. IPSP data was given in [**BGC03, PBD$^+$99, PHT03, PGBZ$^+$06, TWWB02**] for rats, in [**GWGR03**] for ferrets, and in [**TSB98**] for cats. In these publications the change of the inhibitory postsynaptic potential (IPSP) was measured.

Reported IPSC amplitudes range from 8.6 pA [**WGTR$^+$02**] to 32.4 ± 18.0 pA [**BCHS95**] in rat somatosensory and hippocampus, respectively.

Available data of IPSP dynamics in pyramidal neurons mention observed IPSP amplitudes ranging from 0.65 ± 0.44 mV [**TWWB02**] to 1.1 ± 0.8 mV [**BGC03**]. Corresponding rise times range from 1.5 ± 0.7 ms [**BGC03**] ($20-80\%$) to 6.8 ± 2.7 ms ($10-90\%$) [**PBD$^+$99**], respectively. PSP decay times are usually given by the half-width of the potential change and are ranging from 21.5 ± 1.9 ms [**TWWB02**] to 47.2 ± 16.9 ms [**PBD$^+$99**].

For visual cortex in cat (Layer 4) we found in [**THMJS98**] IPSP amplitudes of 0.808 ± 0.448 mV at a half-width of 21.3 ± 7.4 ms.

Available data of IPSP dynamics in inhibitory neuronal targets mention IPSP amplitudes ranging from 1.2 ± 0.7 mV [**PHT03**] to 2.0 mV [**TWWB02**]. Corresponding rise times range from 1.51 ± 0.53 ms [**TSB98**] $(10 - 90\%)$ to 3.9 ms $(10 - 90\%)$ [**TWWB02**], respectively. PSP decay time, given by the half-width of the potential change and are ranging from 6.66 ± 1.24 ms [**TSB98**] to 13.3 ± 3.4 ms [**PHT03**].

Modeling work in [**HT05**] uses GABA currents with a rise times of 1.0 ms and 60.0 ms for $GABA_A$ and $GABA_B$, respectively. The corresponding decay times in their model are 7.0 ms and 200 ms. In [**LRDL06**] only $GABA_A$ currents are used. They use instantaneous rise time (0.0 ms) and a decay time of 6.0 ms.

6.4. Conclusion and Discussion

After reviewing the literature we assembled a set of seemingly reasonable parameters for spike based simulations. The neuronal and synaptic parameters shown in Table 6.1 are the ones we used in Chapter 5 and are the basis for the simulations in Chapter 7.

Given the wide ranges of plausible parameter regimes and the large standard deviations for individual experiments it is impossible to assemble *the* correct set of bioplausible parameters. The data in Section 6.3 suggests that even two neighboring neurons of the same type are likely to use different parameters. We think that neural networks in our brain are likely to have found a way to deal with such vast heterogeneities. It seems therefore desirable to study artificial neural networks that (*i*) are able to deal with this heterogeneity among their computational units [**RHK$^+$11**], and (*ii*) turn out to work even more reliably because of this heterogeneity.

We think that the variability in the instantiation of biological processing units will finally be understood to be a feature, not a bug. In Chapter 7 we will see one way this might be true.

TABLE 6.1. Default LIF parameters used in spiking simulations in Chapters 5 and 7[1]. Plausibility of these parameters are discussed in Section 6.3. The parameters and LIF neurons themselves are described in Section 6.2.

Excitatory Cells

parameter	default	unit
V_{rest}	-65.0	mV
V_{reset}	-65.0	mV
V_{th}	-52.0	mV
τ_{m}	20.0	–
t_{ref}	2.0	ms
ρ_{AMPA}	0.5	–
τ_{AMPA}	1.5[2]	ms
$\tau_{\text{NMDA,rise}}$	10.0[2]	ms
τ_{NMDA}	100.0[2]	ms
τ_{GABA}	5.5 or 11.0[1]	ms

Inhibitory Cells

parameter	default	unit
V_{rest}	-60.0	mV
V_{reset}	-60 or -45[1]	mV
V_{th}	-40.0	mV
τ_{m}	10.0	–
t_{ref}	1.0	ms
ρ_{AMPA}	1.0	–
τ_{AMPA}	1.5	ms
τ_{GABA}	5.5	ms

[1] In Chapter 7 we use the parameters in Table 6.1 to define parameter ranges from which neurons and synapses are assigned values.

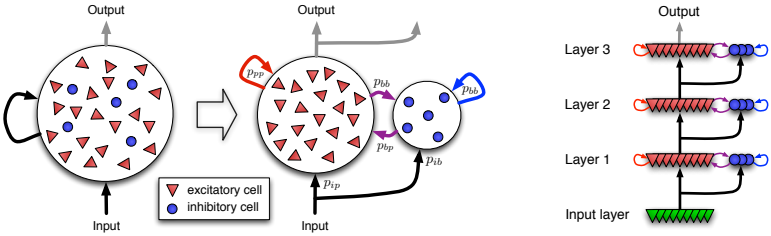
[2] Values for τ_{AMPA}, $\tau_{\text{NMDA,rise}}$, and τ_{NMDA} where chosen to fit the mixed AMPA+NMDA EPSPs reported in [**BGC03**].

CHAPTER 7

Avoiding and Inducing Oscillations in Recurrent Networks of Excitatory and Inhibitory Neurons

7.1. Introduction

Recurrent networks, such as the one shown in Figure 7.1(a), can be found in different places in the central nervous system of many species. It is therefore not very surprising that a plethora of investigations on such networks can be found in the literature [**HRB11, RRWF10, PMBA$^+$09, VA09, KRA08, BSF07, ASLB07, VA05, BDM04,**



(a) **(b)**

FIGURE 7.1. An RCN and a chain of RCNs. **(a)** An *Recurrent Competitive Network* (RCN). The left side shows a diagram of the chaotic distribution of cell bodies in Euclidian brain space. The right side shows the same network, this time with excitatory and inhibitory cells visually separated. Solid arrows symbolize a set of randomly drawn connections between cells of the source and target populations. The number of connections can be controlled by the parameters set $\{p_{ip}, p_{ib}, p_{pp}, p_{pb}, p_{bb}, p_{bp}\}$. **(b)** A feed-forward network containing three RCN layers like the one shown in (a). In green we show a set of input neurons used to feed activity into the first layer.

DGFM03, BW03, Bru00, AB97, BS91]. The canonical microcircuit [**BDM04**], for example, contains several places where RCN-like structures can be found. We expect that these biological architectures exploit very similar dynamics to what we investigate here.

Unfortunately we do not yet understand these circuits well enough to be able to build artificial networks with brain-like data processing abilities. A key problem is that these systems are highly self-referential (recurrent) and cannot easily be tackled analytically. Although a mathematical description of the network dynamics is often possible using coupled differential equations, finding closed-form solutions for these is generally not possible.

Synchrony and Asynchrony in Neural Networks. The spike pattern of a single cell can, depending on the distribution of its inter-spike intervals (ISI), be categorized as having either regular or irregular spiking. Most cortical cells show an ISI distribution that is roughly Poisson-distributed, which makes the individual spikes appear to be generated at "random" points in time [**SK93**].

At the level of neural populations we can distinguish synchronous and asynchronous population spike patterns. This distinction can be made on the basis of temporal proximity of the spikes generated by the members of the cell population.

Proximal cells in cortical networks do not usually fire in tight synchrony, but their firing can be correlated. Although synchronization seems to be an ubiquitous feature in central nervous systems [**OL08, SK08, Buz06, BZN01, LRF99**], data recorded from nearby cells in cortex often does not show synchronized spiking [**SM10, EBK$^+$10, RMV01**]. Such cells might, on a timescale of seconds, be active at similar times without their spike times matching on a timescale of tens of milliseconds or less. The RCN structure shown in Figure 7.1(a) is capable of generating both, synchronous as well as asynchronous spike patterns.

Simulated recurrent networks have the strong tendency to synchronize [**KRA08, Rey03**] and it can be difficult to overcome this effect. The computational primitives of synfire chains [**KRA10, DC06, Izh06, DGA99, Abe82**], for example, are based on the synchronization properties of spiking network models.

Our Contribution. We explore the synchronization properties in two simulated networks of the type shown in Figure 7.1(a). While the first

example network shows reliable synchronization properties, the second is very reliable in desynchronizing even synchronized inputs and generates almost perfect Poisson distributed population ISIs.

This proves the existence of desynchronizing RCNs and shows that they can reliably cope with a wide range of input frequencies and input distributions.

A third example network is similar to the desynchronizing setup we have just mentioned. The difference is that all neurons and synapses contained in the network are no longer perfect copies of each other. Instead we build a *heterogeneous network* where the parameters for each cell and synapse are randomly drawn from some predefined parameter range or parameter distribution. We can show that the third example network, although it consists of very diverse components having different integration and spiking properties, does not qualitatively change its overall dynamics and still behaves very much like its "monocultured" equivalent (the second example network).

As a last result we show that heterogeneous networks do not necessarily desynchronize population spiking. We modify the first example network in the same way we have modified the second example network and show that its synchronization properties are resistant to the diversity of the components in the heterogeneous network.

The dynamics of the example networks show that reliably synchronizing as well as reliably desynchronizing recurrent networks can be built using biologically plausible parameters or parameter ranges.

Related Work. Related work mentioned in Chapter 5 (page 39), is mostly also relevant for this chapter. Mainly the work of Brunel and his collaborators [**HRB11, Bru00, AB97**] motivated us to have a closer look at synchronization and desynchronization in networks of spiking neurons.

Synchronous firing can be important for cortical computations. The effect of weak inputs can be significantly enhanced if they arrive synchronously, so that the EPSPs can combine to reach the threshold before the membrane leakages makes the transmembrane potential decay towards its natural resting state. For weak sensory stimuli it might even be necessary to receive synchronized input to reliably excite cortical cells [**BMS07, BS06, WSFS10**].

7.2. Methods

7.2.1. Model Neurons. Neurons in our simulations are modeled by leaky integrate-and-fire (LIF) neurons [**Tuc88, DA01**]. Details about this model neuron and a biologically plausible parameterization can be found in Chapter 6.

7.2.2. Heterogeneous Populations. Biophysical experiments confirm that individual cells or synapses show huge variances for basically all their measurable parameters. In Chapter 6 we have reviewed the literature for biologically plausible parameters for LIF neurons. The data we collected there strongly supports the claim that neural populations in real cortical tissues are fairly heterogeneous.

In this chapter we will use "heterogeneous populations", which are cell assemblies where each neuron and synapse is *not* a perfect copy of an archetypical blueprint[1]. Instead we define value ranges for many parameters of LIF neurons and the synapse model we use. Every time we instantiate a neuron or synapse to incorporate it in a heterogeneous network we draw random values from these ranges to parameterize the newly created network element. This leads to networks where each element's parameterization is unique.

7.2.3. The Example Networks. All example networks have the principle structure shown in Figure 7.1(b). All results shown here were obtained in networks of 5 layers. Each layer in this feed-forward network is an RCN. See chapter Chapter 5 for a detailed introduction to this type of network.

Example Network 1. Table 7.1, shown below, contains all network parameters needed to build one "monocultured"[2] RCN layer of the first example network. All neuronal and synaptic parameters for this network are collected in Table 6.1 on page 71.

Example Network 2. Table 7.2, shown below, contains all network parameters needed to build one monocultured RCN layer of the second example network. All neuronal and synaptic parameters for this network are collected in Table 6.1 on page 71.

[1]The thesis of my colleague Christoph Krautz [**Kra12**] does also contain interesting work on heterogeneous RCNs.

[2]We use this term for homogeneous network, networks that consists only of cells that parametrized in the exact same way.

TABLE 7.1. The set of parameters to build a monocultured, synchronizing RCN setup containing 1000 excitatory and 250 inhibitory neurons. The given values d denote the ranges of transmission delays (in milliseconds) a synapse draws its value from when created.

Example Network 1: a chain of monocultured RCNs							
n_p	n_b	p_{ip}	p_{ib}	p_{bp}	p_{pb}	p_{pp}	p_{bb}
		0.10	0.10	0.10	0.10	0.10	0.10
1000	250	w_{ip}	w_{ib}	w_{bp}	w_{pb}	w_{pp}	w_{bb}
		3.50	4.00	−3.00	3.00	0.50	−1.00
		d_{ip}	d_{ib}	d_{bp}	d_{pb}	d_{pp}	d_{bb}
		[4,7]	[1,2]	[1,7]	[0.1,2]	[1,0]	[0.1,1]

TABLE 7.2. The set of parameters to build a monocultured, desynchronizing RCN setup containing 1000 excitatory and 250 inhibitory neurons. The given values d denote the ranges of transmission delays (in milliseconds) a synapse draws its value from when created.

Example Network 2: a chain of monocultured RCNs							
n_p	n_b	p_{ip}	p_{ib}	p_{bp}	p_{pb}	p_{pp}	p_{bb}
		0.10	0.10	0.10	0.10	0.10	0.15
1000	250	w_{ip}	w_{ib}	w_{bp}	w_{pb}	w_{pp}	w_{bb}
		2.50	4.00	−2.50	2.00	1.00	−1.00
		d_{ip}	d_{ib}	d_{bp}	d_{pb}	d_{pp}	d_{bb}
		[0.1,10]	[0.1,1.0]	[0.1,1.0]	[0.1,1.0]	[0.1,10]	[0.1,1.0]

Example Network 3. Table 7.3, shown below, contains all network parameters and parameter ranges needed to build one RCN layer of the third example network. All neuronal and synaptic parameters and parameter ranges for this heterogeneous network are collected in Table 7.5.

Example Network 4. Table 7.4, shown below, contains all network parameters and parameter ranges needed to build one RCN layer of the fourth example network. All neuronal and synaptic parameters and parameter ranges for this heterogeneous network are collected in Table 7.5.

TABLE 7.3. The set of parameters to build a heterogeneous, desynchronizing RCN setup containing 1000 excitatory and 250 inhibitory neurons. The given values d denote the ranges of transmission delays (in milliseconds) a synapse draws its value from when created.

Example Network 3: a chain of heterogeneous RCNs							
n_p	n_b	p_{ip}	p_{ib}	p_{bp}	p_{pb}	p_{pp}	p_{bb}
		0.10	0.10	0.10	0.10	0.10	0.15
1000	250	w_{ip}	w_{ib}	w_{bp}	w_{pb}	w_{pp}	w_{bb}
		[1.25,3.75]	[2.0,6.0]	[-3.75,-1.25]	[1.0,3.0]	[0.5,1.5]	[-1.5,-0.5]
		d_{ip}	d_{ib}	d_{bp}	d_{pb}	d_{pp}	d_{bb}
		[0.1,10]	[0.1,1.0]	[0.1,1.0]	[0.1,1.0]	[0.1,10]	[0.1,1.0]

TABLE 7.4. The set of parameters to build a heterogeneous, synchronizing RCN setup containing 1000 excitatory and 250 inhibitory neurons. The given values d denote the ranges of transmission delays (in milliseconds) a synapse draws its value from when created.

Example Network 4: a chain of heterogeneous RCNs							
n_p	n_b	p_{ip}	p_{ib}	p_{bp}	p_{pb}	p_{pp}	p_{bb}
		0.10	0.10	0.10	0.10	0.10	0.10
1000	250	w_{ip}	w_{ib}	w_{bp}	w_{pb}	w_{pp}	w_{bb}
		[1.75,5.25]	[2.0,6.0]	[-4.5,-1.5]	[1.5,4.5]	[0.25,0.75]	[-1.5,-0.5]
		d_{ip}	d_{ib}	d_{bp}	d_{pb}	d_{pp}	d_{bb}
		[4,7]	[1,2]	[1,7]	[0.1,2]	[1,0]	[0.1,1]

7.2.4. Fed Input. External input is fed to the first layer in a chain of individual RCNs. This input can be of various types, ranging from asynchronous, Poisson-like input patterns to highly synchronous population bursts. Figure 7.2 shows three examples of such inputs – each of them is in some way an extreme example.

A useful measure of synchronous or asynchronous population spiking is the *coefficient of variation* (CV), which is defined as the ratio of the standard deviation σ, and the mean μ of the distribution of inter-spike intervals (ISIs) of individual neurons, or in our case of the ISIs between all spike times generated by the entire (input) population (in which case most ISIs are between spikes generated by different neurons).

TABLE 7.5. LIF parameter ranges used in Example Networks 3 & 4. Plausibility of these parameters is discussed in Section 6.3. The parameters and LIF neurons themselves are described in Section 6.2.

Excitatory Cells

parameter		unit
V_{rest}	[-67.5, -62.5]	mV
V_{reset}	[-67.5, -62.5]	mV
V_{th}	-52.0	mV
τ_{m}	[17.5, 22.5]	–
t_{ref}	[1.5, 2.5]	ms
ρ_{AMPA}	[0.25, 0.75]	–
τ_{AMPA}	1.5	ms
$\tau_{\text{NMDA,rise}}$	10.0	ms
τ_{NMDA}	100.0	ms
τ_{GABA}	11.0	ms

Inhibitory Cells

parameter		unit
V_{rest}	[-62.5, -57.5]	mV
V_{reset}	[-47.5, -42.5]	mV
V_{th}	-40.0	mV
τ_{m}	[7.5, 12.5]	–
t_{ref}	[0.75, 1.25]	ms
ρ_{AMPA}	1.0	–
τ_{AMPA}	1.5	ms
τ_{GABA}	5.5	ms

$$\text{CV} = \frac{\sigma_{\text{ISI}}}{\mu_{\text{ISI}}} \tag{7.1}$$

Figure 7.2 also gives some clues about the values we would expect for the three kinds of inputs. Since very uniformly distributed spike times, as in the interleaved example of Figure 7.2(a), have a very small variance ($\sigma_{\text{ISI}} < \epsilon$), the CV value for such inputs will be very small as well.

If, in contrast to that, the spike times are synchronized, as in Figure 7.2(c), we will have many very small ISIs between the synchronous spike times, and then a fairly large one between such synchronous events. This leads to a relatively large σ_{ISI} compared to the small μ_{ISI}.

For Poisson distributed ISIs the CV is precisely 1. This is because the standard deviation for a Poisson distribution coincides with its mean, which makes $\sigma_{\text{ISI}} = \mu_{\text{ISI}}$.

7.3. Results

As described in Section 7.2.4, we tested the synchronization properties of the four sample networks by feeding them the three types of input patterns shown in Figure 7.2. These inputs were chosen to be extreme instances in terms of their synchronization statistics.

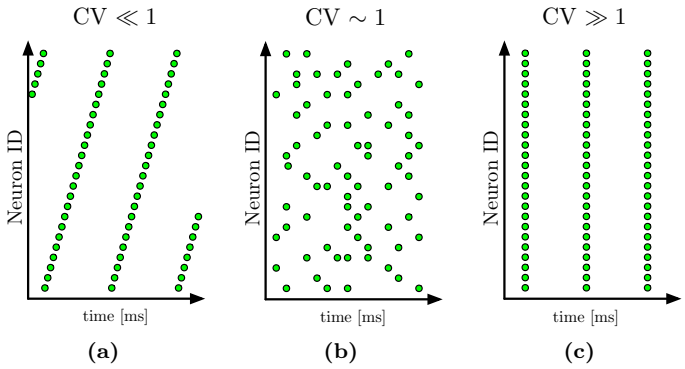


FIGURE 7.2. Three types of inputs that can be fed to the simulated networks. The desired feature of the network (synchrony or desynchronization) should be robust enough to be able to deal with inputs of all three kinds. **(a)** Spike events are uniformly interleaved. The ISI variance between all consecutive spikes is almost zero, hence the CV value (coefficient of variation) of the entire input population is much less than 1. **(b)** Spike events are roughly Poisson distributed, hence the CV value is close to 1. **(c)** Spike events are almost synchronous. The ISI variance is a multiple of the mean time between all consecutive spike events, hence the CV value is much bigger than 1.

In the introduction to this chapter we mentioned that synchronous as well as asynchronous population spiking seems to be relevant for neural computation. The dynamics shown in the example networks show that reliably synchronizing as well as reliably desynchronizing recurrent networks can be built using biologically plausible parameters or parameter ranges.

7.3.1. Example Network 1: Synchronized Spiking in a Homogeneous Network. The first example network is built by using a fixed set of parameters for all cells in the entire network[3].

In Figure 7.3 we show how Example Network 1 deals with a wide range of different inputs. Each subfigure contains data for (*i*) uniformly interleaved, (*ii*) Poisson, or (*iii*) synchronized input patterns.

[3]For excitatory and inhibitory cells we use different parameters, but each excitatory (resp. inhibitory) cell is created with precisely the same parameterization.

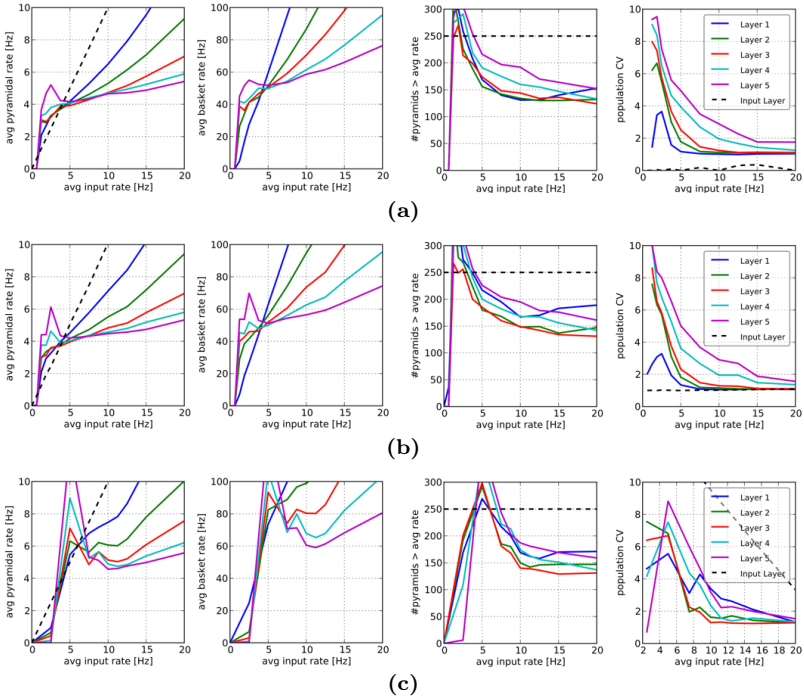


FIGURE 7.3. Simulation results for Example Network 1. The plots are explained in detail in Section 7.3.1. The last plot in each row shows that this network reliably generates synchronized population outputs. Raster plots of the raw spike-data can be found in Figure 7.4. **(a)** For uniformly interleaved input as in Figure 7.2(a). **(b)** For Poisson input as in Figure 7.2(b). **(c)** For synchronized input as in Figure 7.2(c).

For each type of input we set 250 input neurons to generate a fixed spike rate for several seconds and recorded the output spike rates of all pyramidal cells in all 5 layers of the network. We repeated this procedure for different spike rates ranging from 0Hz up to 80Hz. Since the input layer contains 1000 neurons in total this corresponds to average input rates of 0Hz to 20Hz (because only one quarter of all input neurons generate this rate).

The x-axes in all plots in Figure 7.3 show the average rate of the input layer. The y-axes, in order of appearance from left to right, show (i)

the average rate of the pyramidal population for all 5 layers and the input layer, (*ii*) the average rate of the basket cell population for all 5 layers, (*iii*) the number of pyramidal cells in each layer that is active with a higher rate than the average rate of all pyramidal cells of that layer, and (*iv*) the population CV value for the pyramidal population in each of the 5 layers of the network.

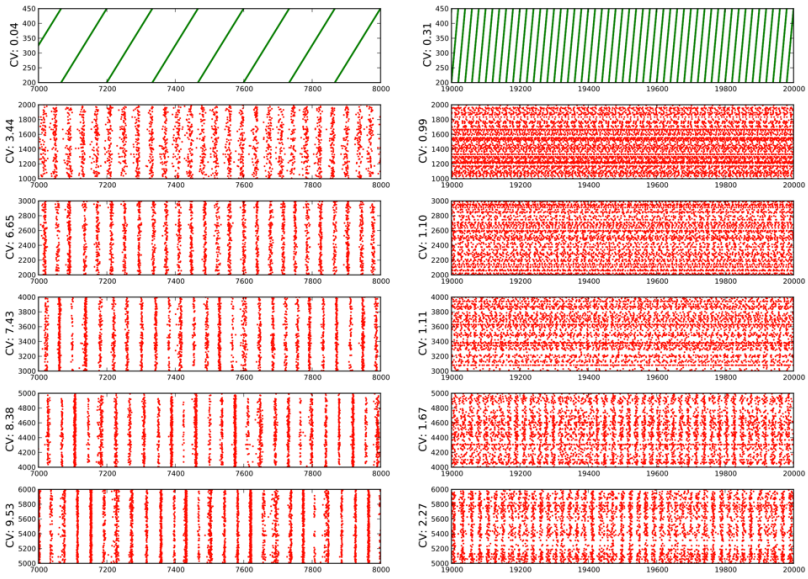


FIGURE 7.4. Raster plots for two rates of uniformly distributed input fed to Example Network 1. **(a)** For 7.5Hz, which is an average of slightly below 2Hz. **(b)** For 40.0Hz, which is an average of 10.0Hz.

Figure 7.4 shows raster plots of raw spike-data for two input rates. One fairly low input rate of $x = 7.5$ Hz (corresponding to an average rate of ≈ 2 Hz), and a higher rate $x = 40$ Hz (corresponding to an average rate of 10 Hz). Raster plots of higher input rates do not look considerably different.

We show the raster plots for uniformly interleaved input ISIs, since it is the most challenging one for the network to synchronize.

The data shown in Figures 7.3 and 7.4 shows that Example Network 1 creates synchronized population spiking for all fed inputs. The measured CV values become lower for higher input rates (last column in Figures 7.3), but they do increase, and the raster plots in Figure 7.4 show that the population ISIs are still far away from being Poisson distributed.

7.3.2. Example Network 2: Desynchronized Spiking in a Homogeneous Network. The second example network is again built by using a fixed set of parameters for all cells in the entire network. We show the same kind of data we have shown for Example Network 1 in Figures 7.5 and 7.6.

The input classes as well as the rates of the input layer in Figures 7.5 are as explained in Section 7.3.1 for Example Network 1.

Figure 7.6 shows raster plots of raw spike-data for two input rates $x = 30.0$ Hz (corresponding to an average rate of 7.5 Hz), and $x = 50$ Hz (corresponding to an average rate of 12.5 Hz). Raster plots of higher input rates do not look considerably different.

We show the raster plots for synchronized spiking input, since it is the most challenging for the network to desynchronize.

The data shown in Figures 7.5 and 7.6 shows that Example Network 2 creates desynchronized population spiking for all fed inputs. The measured CV values are very close to 1 for basically all input rates (last column in Figure 7.5). Also the raster plots in Figure 7.6 show that the population ISIs in later layers are evenly distributed, not clustering much in time.

7.3.3. Example Network 3: Desynchronized Spiking With Heterogeneity. The third example network is similar to Example Network 2. The biggest difference is that the network no longer consists of cells and synapses that use the same parameter values, but each neuron and each synapse draws its parameters uniformly at random from predefined parameter ranges (see Section 7.2).

Again we show the same kind of data we have shown for Example Networks 1 & 2 in Figures 7.7 and 7.8.

The input classes as well as the fed input ranges in Figures 7.5 are just as explained in Section 7.3.1 for Example Network 1.

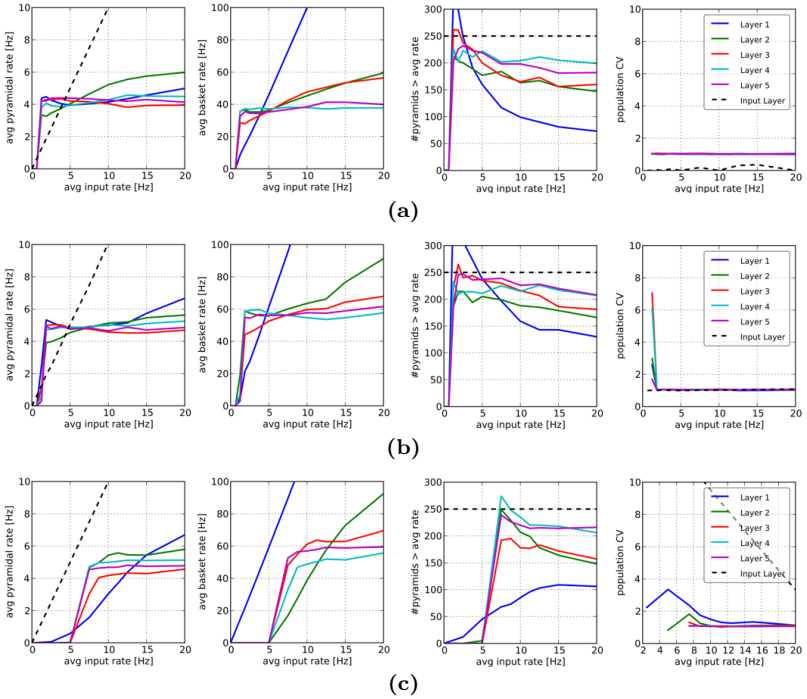


FIGURE 7.5. Simulation results for monocultured Example Network 2. The plots are explained in Section 7.3.1 and discussed in Section 7.3.2. The last plot in each row shows that this network reliably generates desynchronized, Poisson-like population outputs. Raster plots of raw spike-data can be found in Figure 7.4. **(a)** For uniformly interleaved input as in Figure 7.2(a). **(b)** For Poisson input as in Figure 7.2(b). **(c)** For synchronized input as in Figure 7.2(c).

Figure 7.8 shows raster plots of raw spike-data for two input rates: 35.0 Hz (corresponding to an average rate of 8.25 Hz), and 50 Hz (corresponding to an average rate of 12.5 Hz). Raster plots of higher input rates do not look considerably different.

We show the raster plots for synchronized spiking inputs, since it is the most challenging for the network to desynchronize.

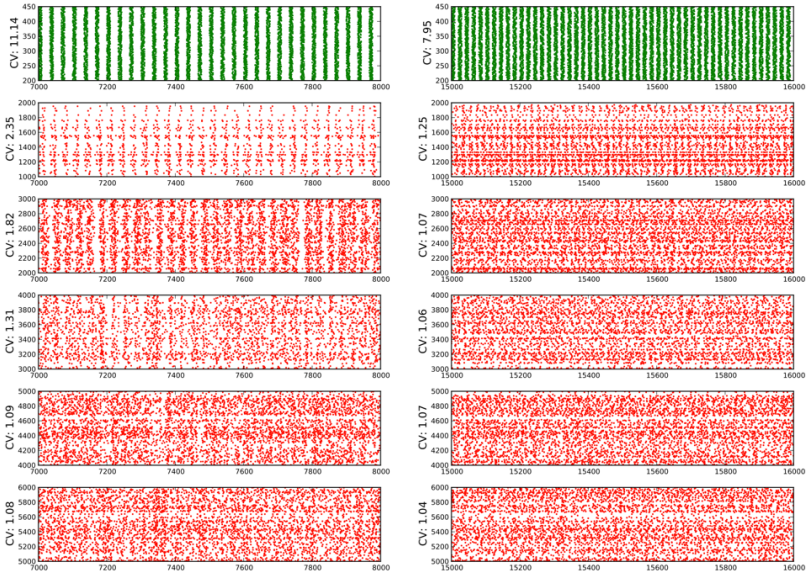


FIGURE 7.6. Raster plots for two rates of synchronous input fed to Example Network 2. **(a)** For 30.0Hz, which is an average of 7.5Hz. **(b)** For 50.0Hz, which is an average of 12.5Hz.

The data shown in Figures 7.7 and 7.8 shows that Example Network 3 creates desynchronized population spiking for all fed inputs. The measured CV values are very close to 1 for basically all input rates (last column in Figure 7.7). Also the raster plots in Figure 7.8 show that the population ISIs in later layers are evenly distributed, not clustering much in time.

7.3.4. Example Network 4: Synchronized Spiking With Heterogeneity. The fourth and last example network is similar to Example Network 1 of Section 7.3.1. The biggest difference is that the network no longer consists of cells and synapses that use the same parameter values, but each neuron and each synapse draws its parameters uniformly at random from predefined parameter ranges (see Section 7.2).

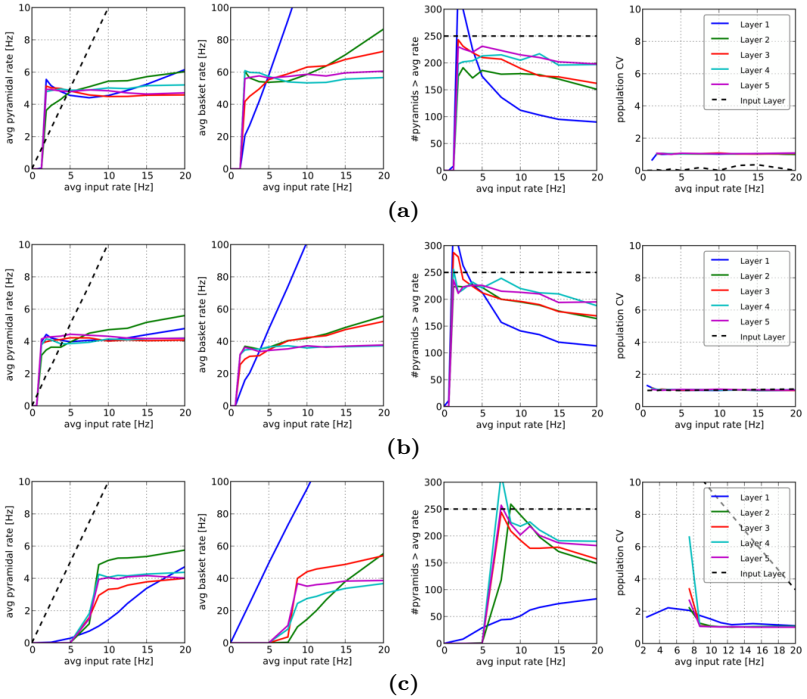


FIGURE 7.7. Simulation results for the heterogeneous Example Network 3. The plots are explained in Section 7.3.1 and discussed in Section 7.3.3. The last plot in each row shows that this network reliably generates desynchronized, Poisson-like population outputs. Raster plots of raw spike-data are shown in Figure 7.8. **(a)** For uniformly interleaved input as in Figure 7.2(a). **(b)** For Poisson input as in Figure 7.2(b). **(c)** For synchronized input as in Figure 7.2(c).

Again we show the same kind of data we have shown for Example Networks 1-3 in Figures 7.9 and 7.10.

The input classes as well as the fed input ranges in Figures 7.7 are just as explained in Section 7.3.1 for Example Network 1.

The data shown in Figures 7.10 and 7.8 shows that Example Network 4 creates synchronized population spiking for all fed inputs. The measured CV values become lower for higher input rates (last column in Figures 7.7), but they do increase, and the raster plots in Figure 7.8

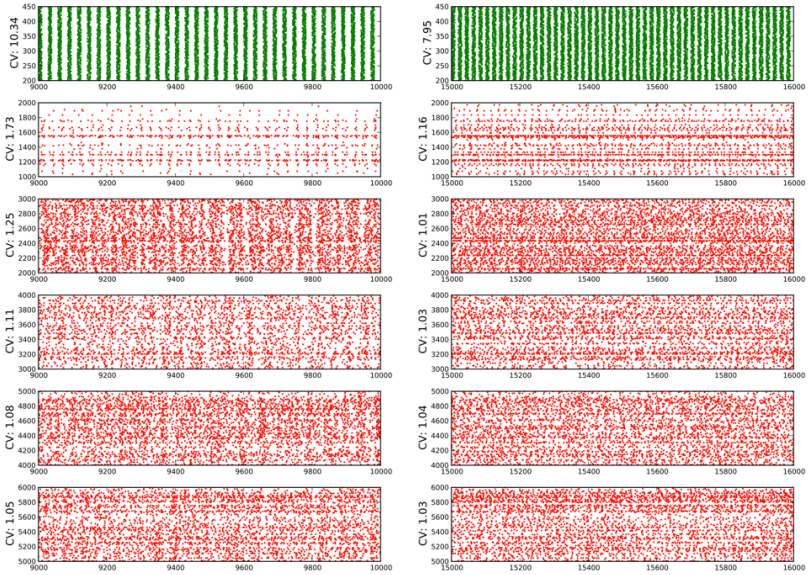


FIGURE 7.8. Raster plots for two rates of synchronous input fed to Example Network 3. **(a)** For 35.0Hz, which is an average of 8.75Hz. **(b)** For 50.0Hz, which is an average of 12.5Hz.

show that the population ISIs are still far away from being Poisson distributed.

Figure 7.10 shows raster plots of raw spike-data for two input rates: 7.5 Hz (corresponding to an average rate of ≈ 2 Hz), and 40 Hz (corresponding to an average rate of 10 Hz). Raster plots of higher input rates show similar behaviour.

Like for Example Network 1 we decided to show the raster plots for uniformly interleaved input ISIs, because it is the most challenging for the network to synchronize.

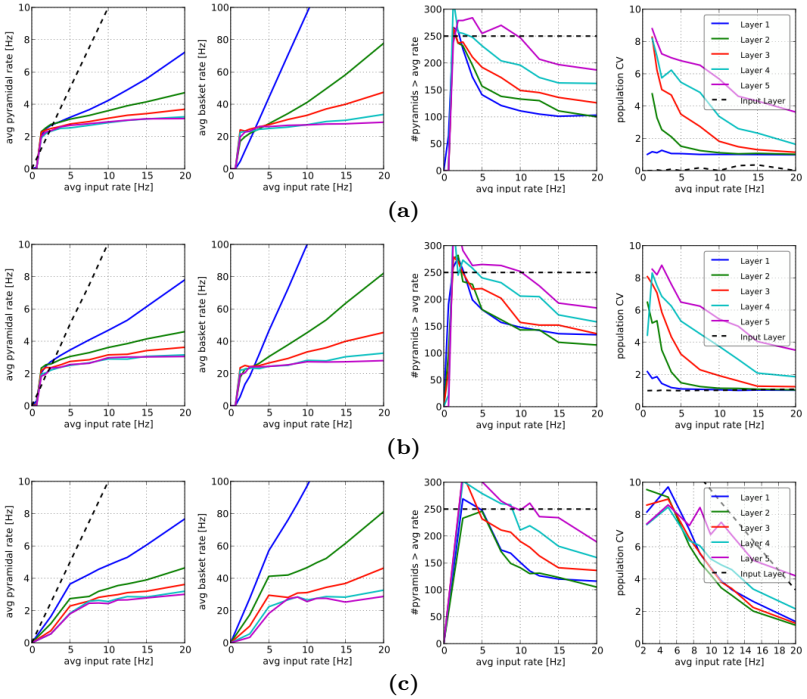


FIGURE 7.9. Simulation results for the heterogeneous Example Network 4. The plots are explained in Section 7.3.1 and discussed in Section 7.3.4. The last plot in each row shows that this network reliably generates synchronized population outputs. Raster plots of raw spike-data are shown in Figure 7.10. **(a)** For uniformly interleaved input as in Figure 7.2(a). **(b)** For Poisson input as in Figure 7.2(b). **(c)** For synchronized input as in Figure 7.2(c).

7.4. Discussion

Synchronous as well as asynchronous spiking seems to be relevant for neural computation (see Section 7.1). In this chapter we showed examples for synchronizing as well as desynchronizing layered feed-forward networks, where each layer was a very general recurrent network of the type we have introduced in Chapter 5.

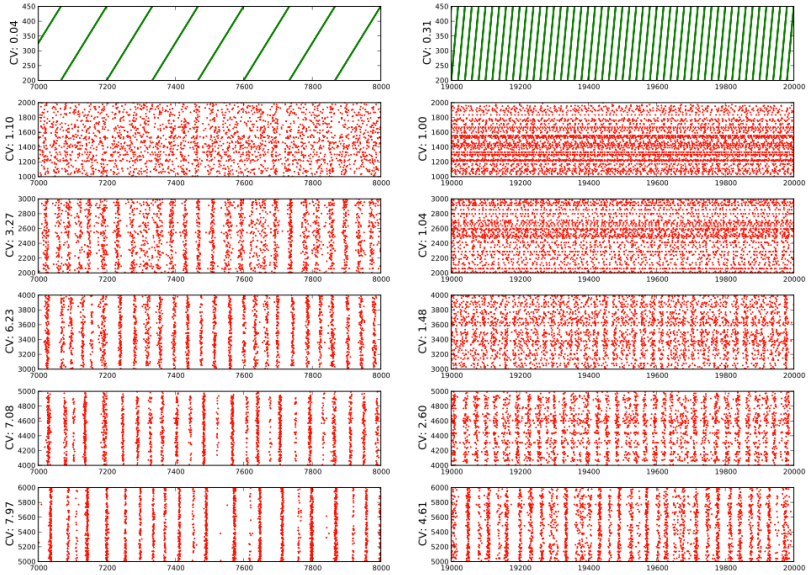


FIGURE 7.10. Raster plots for two rates of uniformly distributed input fed to Example Network 4. **(a)** For 7.5Hz, which is an average of slightly below 2Hz. **(b)** For 40.0Hz, which is an average of 10.0Hz.

In previous work by Brunel and others, see for example [**AB97, Bru00, HRB11**], different spike regimes for RCN networks were analyzed. These publications provide bifurcation diagrams showing how their networks transition between these regimes. To avoid confusion between their findings and the results presented here, we hint at two important differences. (*i*) The networks studied in [**AB97, Bru00, HRB11**] consist of homogeneous populations, i.e. all neurons in them are identical copies, only differing in the connection to their peers. Exchanging these homogeneous populations by heterogeneous cell assemblies would lead to spike patterns which they would classify as asynchronous. This is because of (*ii*), a difference in the sematics of the terms "synchronous" and "asynchronous". These are often used without defining exaclty how close in time spikes have to be in order to be considered synchronous. Often only spikes within very short time windows, in the order of millisecond,

are considered to be synchronous. In [**Bru00**], for example, large areas in their bifurcation diagrams are labeled asynchronous, although spike patterns of such networks show irregular oscillation patterns (see Figure 8(c) in [**Bru00**]), which others might consider as being synchronous.

7.4.1. The Coefficient of Variation for Single Cells. In this chapter we used the coefficient of variation (CV), see Equation 7.1, to describe the synchrony and asynchrony for entire cell populations.

This statistical measure is often used to quantify the regularity or irregularity of spike patterns of single cells. In [**SK93**], for example, one can find CV values for recordings from single cells in cortical areas V1 and MT of awake behaving macaque monkeys.

In Figures 7.11 and 7.12 we plot the distribution of single cell CV values measured in 3 of the simulations described in the results section of this chapter. The measured values are in agreement with the data reported in [**SK93**].

7.4.2. Comparision of Monocultured and Heterogeneous Networks. It might be surprising that Example Networks 1 & 4 as well as Example Networks 2 & 3 generate such similar output patterns and output statistics. We are currently working on a more detailed comparison between monocultured and heterogeneous RCNs, but we can already say that RCNs cope very well with "unreliable", heterogeneous components.

When comparing Figure 7.3 and Figure 7.9 we can even see that the heterogeneous network seems to show smoother input response curves (leftmost column; see also Figure 7.10).

We think that heterogenous network models can not only work reliable, but that heterogeneity can often be a feature that can make networks more robust and reliable.

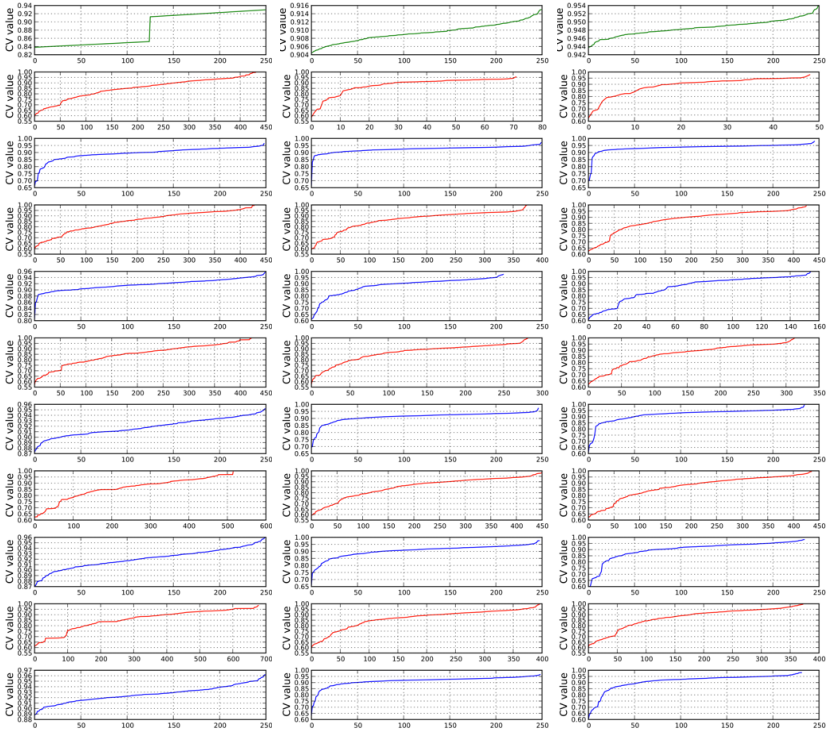


FIGURE 7.11. Single cell CV distributions at low input rate. The measured values are in agreement with the data reported in [**SK93**]. **(a-c)** Top row contains CV distributions of the input layer (green), further below each pair of rows contains the data of RNC layers 1 to 5. Pyramid data is shown in red, basket cell data in blue. **(a)** Shows the CV values for all active cells in Example Network 1. The input rate of the 250 active input neurons is 7.5Hz, which is an average activity of slightly below 2Hz. **(b)** Shows the CV values for all active cells in Example Network 2. The input rate of the 250 active input neurons is 30Hz, which is an average activity of 7.5Hz. **(c)** Shows the CV values for all active cells in Example Network 3. The input rate of the 250 active input neurons is 35Hz, which is an average activity of 8.75Hz.

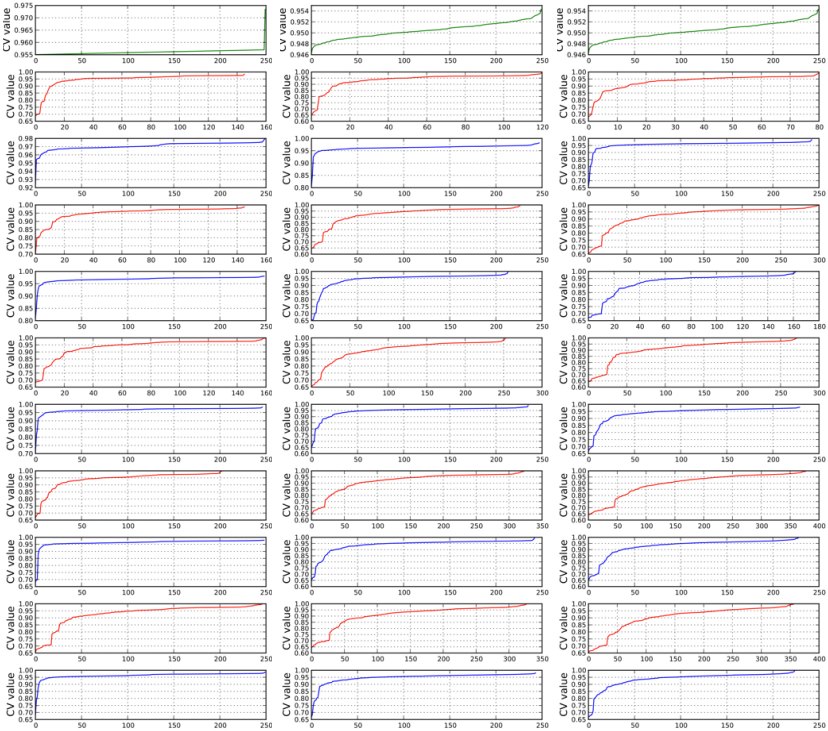


FIGURE 7.12. Single cell CV distributions at slightly higher input rate than in Figure 7.11. The measured values are in agreement with the data reported in **[SK93]**. **(a-c)** Top row contains CV distributions of the input layer (green), further below each pair of rows contains the data of RNC layers 1 to 5. Pyramid data is shown in red, basket cell data in blue. **(a)** Shows the CV values for all active cells in Example Network 1. The input rate of the 250 active input neurons is 40Hz, which is an average activity of 10Hz. **(b)** Shows the CV values for all active cells in Example Network 2. The input rate of the 250 active input neurons is 50Hz, which is an average activity of 12.5Hz. **(c)** Shows the CV values for all active cells in Example Network 3. The input rate of the 250 active input neurons is 35Hz, which is an average activity of 12.5Hz.

CHAPTER 8

Rate Coded RCNs Using Siegert neurons

8.1. Introduction to the Misery of Simulations

Chapters 5 and 7 introduced and investigated spiking Recurrent Competitive Networks (RCNs). Chapter 6 reviewed the literature on biologically plausible parameter ranges for the RCN models we have examined. The simulations in Chapters 5 and 7 were based on leaky integrate-and-fire (LIF) neurons that made use of fairly detailed synaptic dynamics.

Such spiking simulations use detailed models of biological neurons and synapses, and with this comes a tremendous drawback in terms of simulation time. Although we have used well established, state of the art simulation tools, pushed all self-developed algorithms to maximum performance, and used fast multi-core computers, simulation times hardly permit building larger networks consisting of two or more RCNs as interacting modules. But even if further hard- and software enhancements would make it possible to scale our simulations up by an order of magnitude, every attempt to study learning processes in such networks (as opposed to just network dynamics) would still need an almost unbearable amount of simulation time.

Let us consider a short example. Regardless of the time to build the network, the simulation of 1 second of simulated time might on average take 5 seconds on a purely hypothetical system. If our simulated network would like to learn something by seeing a training data set that consists of say 200 data points that are each presented 5 times during learning, the simulation of this process would take 5000 seconds, which is roughly 1 hour and 20 minutes. Even if this seems to be an acceptable duration at first, the problem is that we do not only want to run an already existing neural system – we want to find one. Although we have a concise idea about the way it should and will work, we will most likely have to iterate the simulation multiple times to work out the details.

If it takes 1 hour and 20 minutes every time one wants to check the influence of even tiny changes to the system, you had better already have a good publication record, because your next one might take a while.

There are several ways out of this misery. We will pursue one solution in this chapter.

8.1.1. How We Make Simulations Fast. Let us quickly collect some possible ways out of the "simulation misery".

Theoretical Analysis. The simulation misery is immediately void if we can get rid of the necessity for simulations in the first place. Unfortunately, a thorough theoretical analysis of dynamic, self-referential systems is often infeasible.

Supercomputer. We could leave our multi-core hosts and GPUs behind us and use clusters or supercomputers to simulate our state of the art neural networks. Supercomputer cycles can be affordable and are available for many scientific projects. Unfortunately all algorithms have to be developed and tuned for such architectures and not every computational neurobiologist can or wants to invest the necessary time to develop for supercomputers. Dependency on the availability of supercomputing time for running research projects and the little influence many people most likely have on the provided infrastructure might be an additional problem.

Special-purpose Hardware. Some very interesting alternatives come from the neuromorphic community [**IH11, RGD+11, BPV+11**]. Special-purpose hardware currently offers different philosophies, but all of these paradigms promise to solve simulation issues regarding simulation time and network size.

We think that this movement will impact the field significantly. Currently it is unfortunately not easy to (*i*) get such hardware prototypes in your own lab, and (*ii*) get it to do exactly what you want without (or even with) the help of an expert.

Collaboration with the developers of a suitable neuromorphic type of hardware seems to be the most feasible solution for now. For this to be successful, however, it might be necessary to already have a very detailed idea and a bunch of preliminary results that prove that the suggested network and its dynamics are understood and under control.

Higher Level of Abstraction. Finding a suitable, higher level of abstraction is in general a difficult problem that we can not always hope to solve. This approach is possible whenever (*i*) the system of interest is suitable to be lifted to a higher level of abstraction, (*ii*) we happen to know enough about it to understand how this can be done, and (*ii*) the dynamics of the abstracted system remain similar enough[1] to the dynamics of the original system. In the case that both the mentioned conditions hold true, the proposed abstraction has the potential to offer complexity and performance improvements.

For the simulations of interest in the remaining chapters of this thesis it turns out to be possible to find a suitable higher level of abstraction. In preparation for Chapters 9 and 10 we will introduce the necessary abstractions here. Further we will argue that using these models does not fundamentally alter the observed network dynamics.

8.2. Siegert neurons: a Reasonable LIF Abstraction

The abstract neuron model we are going to use is not new [**RBH$^+$11, OB11, Ric77, Sie51**]. Burkitt, only some years ago, published a review paper describing how leaky integrate-and-fire (LIF) neurons respond to Poisson distributed, spiking inputs [**Bur06**].

His analysis makes use of results like the ones by Siegert [**Sie51**] and Ricciardi [**Ric77**]. Siegert's seminal achievement was the mathematical analysis of an Ornstein-Uhlenbeck process with boundary conditions [**Sie51**], which was subsequently used to describe the sub-threshold dynamics of LIF-neurons when receiving many independent Poisson spike trains. (The sub-threshold dynamics of LIF-neurons can be seen as being a continuous random walk with a drift, parametrized by τ_m, and boundary conditions at V_{rest} and V_{th}.)

These mathematical advances made it possible to describe the evolution of the membrane potential V_m precisely enough to compute the number of times V_m 'hits' the threshold potential V_{th}. This value is effectively the neuron's firing rate λ.

Figure 8.1 shows the necessary evaluation sequence in order to compute the output firing rate of a current based LIF-neuron. We denote the shown module as either "Siegert node" or "Siegert neuron".

[1] What "similar enough" actually means depends very much on the details of a concrete abstraction candidate.

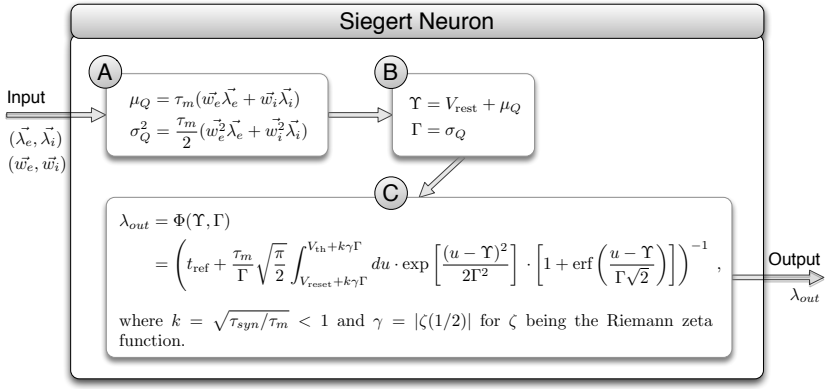


FIGURE 8.1. The Siegert node. The diagram shows the necessary steps for a Siegert neuron to compute its output rate λ_{out}. It receives excitatory and inhibitory inputs where $\vec{\lambda_e}$ ($\vec{\lambda_i}$) are incoming excitatory (inhibitory) rates and $\vec{w_e}$ ($\vec{w_i}$) are the corresponding synaptic weights. The formulas in (A), (B), and (C) are sequentially evaluated, where τ_m, V_{rest}, V_{reset}, V_{th}, and t_{ref} are the same parameters used for leaky integrate-and-fire (LIF) neurons (see Chapter 6). The interpretation of synaptic weights is also analogous to Chapter 6. Note that $\vec{v^2}$ denotes the element-wise square $(v_1^2, v_2^2, \ldots, v_{|\vec{v}|}^2)^T$, and $\vec{w}\vec{\lambda}$ is the vector dot product of $\vec{w}$ and $\vec{\lambda}$.

For conductance based LIF-neurons it is possible to deduce a Siegert-style formalization as well. In this case the formulas in Step 2 of Figure 8.1 become a bit more involved. Although we sometimes use the conductance based version in our research, we will not use it in this thesis. A detailed overview of the history as well as the mathematical analysis of current and conductance based LIF neurons can be found in the doctoral thesis of Krautz [**Kra12**]. We suggest that interested readers start their journey there.

8.2.1. From Single Cells to Entire Networks. Figure 8.1 gives us a formula that can be use to compute how an LIF neuron responds to many independent Poisson spike trains. It gives us the opportunity to

compute the output spike rate of an LIF neuron without any need of simulating one.

Predicting the activity of a singe LIF neuron is of course not very interesting. Simulating a single LIF neuron is so fast that there is no need to use a Siegert neuron to replace the simulation. However, if we would build an entire network consisting of interconnected Siegert neurons, the speed advantage of evaluating only the equations shown in Figure 8.1 instead of doing the LIF network simulation has the potential to save a significant amount of simulation time.

In this thesis we will build an RCN built from Siegert neurons. After building a network we want to check whether the Siegert-simulation leads to results comparable to the corresponding spiking simulation. The Siegert formula describes the activity of single cells very well, but without any notion of individual spike times the Siegert-network can clearly not replicate its LIF equivalent in all its fine details.

In Sections 8.2.2 and 8.3 we compare the results of a Siegert-RCN with the ones obtained using spike based RCNs (LIF-RCNs). Although we will see that the abstraction, as expected, does not lead to the exact same network activations, all the basic properties of RCNs (as described in Chapter 5) are retained.

8.2.2. Building and Evaluating Siegert-RCNs. An RCN built from Siegert neurons is in some ways fundamentally different from the same network using LIF neurons. LIF simulations need a globally consistent notion of time. The communication between LIF neurons is based on discrete spike events and their precise timing is in general very important.

Siegert neurons, on the other hand, live on a higher level of abstraction where time is no longer explicitly modeled. The communication between Siegert nodes is based on spike rates and not individual spike times.

The instantiation of an RCN using either of the two neuron models is of course not a problem. Evaluating the network (running the simulation) is very different. While we usually use NEST [**EHM08, DG02**] for spiking simulations, we implemented a simulation strategy for Siegert-Networks ourselves.

Function **EvaluateNetwork** (*network*, *input_rates*, *input_weights*) on page 98 shows how an arbitrary network of Siegert nodes is evaluated.

In subsequent chapters we will extend this algorithm, but the basic ideas will remain unchanged.

Function **EvaluateNetwork** (*network*, *input_rates*, *input_weights*):

```
1:  /* we perform k_upd iterated evaluations */
2:  for step = 1 → k_upd do
3:    /* collect current inputs for all neurons n */
4:    for each n in network.neurons do
5:      n.inputRates.append(input_rates(n))
6:      n.inputWeights.append(input_weights(n))
7:      for each (sender,weight) in n.incoming_connections do
8:        n.inputRates.append(sender.rate)
9:        n.inputWeights.append(weight)
10:     end for
11:   end for
12:   /* compute new output firing rates */
13:   for each n in network.neurons do
14:     newRate = Siegert(n.inputRates, n.inputWeights)
15:     n.rate = (1 − τ_rate) · n.rate + τ_rate · newRate
16:   end for
17:   /* reset values for next round of input */
18:   for each n in network.neurons do
19:     empty lists n.inputRates, n.inputWeights
20:   end for
21: end for
```

In the following we will provide some additional explanations that might help readers to better understand the pseudo-code description:

Line 2-21 : The main loop. Each cycle updates the entire network. We usually use a value of $k_{\text{upd}} = 10$.

Line 4-11 : Here we collect for each neuron a list of all rates and a list of corresponding weights of all inputs to this neuron, which encodes the current dendritic input. External inputs are given by the parameters *input_rates* and *input_weights*.

Line 13-16 : This code-block uses the dendritic input collected in lines 4-11 to compute the next output rate for each neuron in the given network.

Line 14 **:** Here the Function **Siegert**(...) is called in order to compute the output rate of an LIF neuron receiving the collected input rates with the corresponding synaptic weights. See Figure 8.1 for a description of what this function computes. See the thesis of Chrisoph Krautz [**Kra12**] for a profound summary of the mathematical deduction of these formulas.

Line 15 **:** Here, finally, we update the output rate of neuron n. The parameter τ_{rate} determines how quickly we approach $newRate$, the firing rate computed in line 20. In order to avoid oscillations we usually use values for τ_{rate} between 0.25 and 0.5.

Line 18**-**20 **:** This code-block resets all temporal variables. This has to be done before the main loop can be re-iterated.

8.3. Results: Siegert-RCNs vs. LIF-RCNs

In the previous section we have essentially given a description for how to build arbitrary networks consisting of Siegert neurons and how such networks can be evaluated.

Our main interest in the context of this thesis is to build RCNs as we have introduced them in Chapter 5. In this section we want to compare the dynamics of RCNs built from LIF-neurons with the dynamics of RCNs that consist of Siegert nodes.

Figure 8.2 shows both, the activation of an LIF-RCN (gray) as well as the activation of a Siegert-RCN (green). The simulated activities are very similar, showing us that the abstraction from a spiking model to a rate-coded Siegert-network can lead to almost identical results.

Unfortunately not every set of RCN parameters leads to such a tight fit. Note that the network in Figure 8.2 consisted of 3000 excitatory neurons, hence being even larger then the RCN networks discussed in Chapters 5 or 7.

Larger networks often give better matching results because random fluctuations are smaller. Not only the network size matters: if a set of parameters for a certain size leads to well matching results, the activation of Siegert-RCNs and LIF-RCNs can be significantly less similar for other, seemingly similar parameter sets.

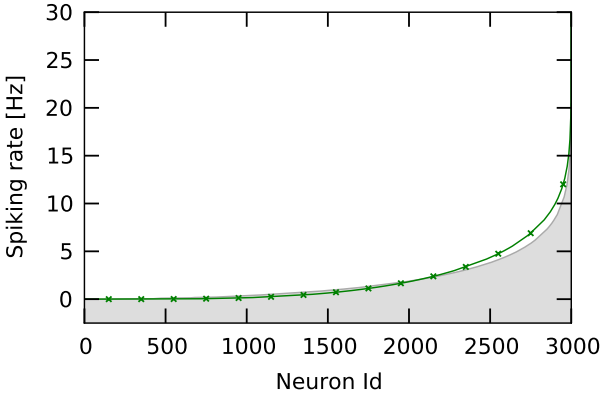
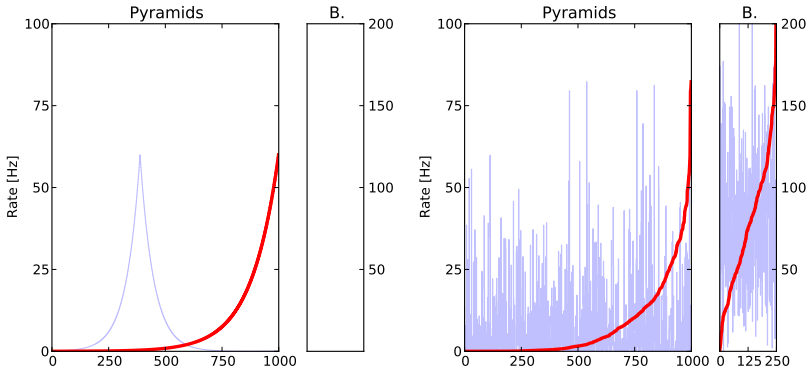


FIGURE 8.2. Activation of the excitatory subpopulation of an LIF-RCN and an equivalent Siegert-RCN. The used network is described in detail in the Thesis of Christoph Krautz [**Kra12**]. The gray area shows the sorted activities of the RCN simulated using LIF neurons (see Chapter 5). In green we have plotted the sorted activation profile of an equivalent Siegert-RCN evaluated using the method described in Section 8.2. The activation profile of these two simulations is very similar for a wide range of possible inputs.

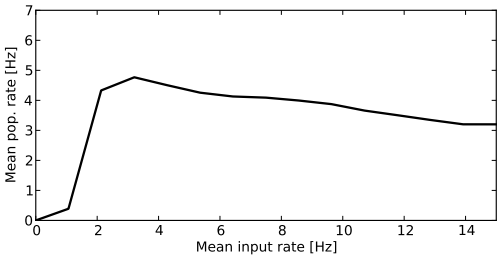
One of the main reasons for mismatching results are synchrony issues (see also Chapter 7). While synchronization of spike times is a fundamental feature in spiking networks, a Siegert-network operates on a level of abstraction that cannot capture such synchronization aspects.

An ongoing research project by the author and his collaborators is to better understand and quantify the reasons for mismatching network activities in spiking networks and isomorphic networks of Siegert neurons.

Does the Shape of the Activation Profile Really Matter? In full generality the only possible answer is of course "YES". However, when looking at specific network tasks or learning tasks the answer might be a different one.



(a)



(b)

FIGURE 8.3. RCN properties are retained in Siegert-RCNs. **(a)** Sparse responses to inputs. The left half of the plot shows the activity of the input layer, the right half the activity of the Siegert-RCN itself. Individual rates of the 1000 pyramids and the 250 baskets is shown in light blue. The bold, red curve shows the same activation data sorted ascending. The figure was created using the RCN parameters given in Table 5.1. The input was chosen to be the same as in Figure 5.2. **(b)** High dynamic range due to feed-forward inhibition [**PMBA$^+$09**] in a network of Siegert nodes. As we have shown in Figure 5.4 for spike based RCNs, excitation and inhibition in Siegert-RCNs can also be balanced such that a large range of input patterns leads to similar total output rates.

In Chapters 9 and 10 we will see that it is possible (maybe even desirable) to build systems that do not care about the precise activation of the nodes in its RCN-layers. In these chapters we will see that, as long as the RCN dynamics described in Chapter 5 are retained, the network adapts its own activation patterns to the ones fed to it from the outside. Nothing about this adaption or learning mechanism prefers one RCN activation to a slightly different one, as long as the response to external input leads to sparse activation and feed-forward inhibition as in Figure 5.2.

RCN properties do not change when using Siegert nodes. We instantiate a Siegert-RCN using the parameters of Table 5.1 on page 43. Using this Siegert-equivalent of the spiking network used in Chapter 5 we can show that all properties of RCNs are well conserved.

Figure 8.3 shows that, although the activation of single units is different from e.g. the one in Figure 5.2, the network as a whole follows the same types of dynamics as spiking RCNs. Although it is important to keep in mind that although switching from LIF-neurons to Siegert neurons or vice versa will usually make a difference for individual nodes, the overall picture does not change.

The networks we are interested in are usually not sensitive to the activity of single nodes or the precise average rate of simulated populations of neurons. One of the reasons why the proposed type of network is interesting is its robustness to such variabilities.

Such robust phenomena could easily be chosen by evolution to play a significant role in the type of computations performed in our brains.

8.4. Discussion: Is there Potential for Siegert Nodes?

We think that the development of spiking neural networks will benefit from the possibility to run quick, prototypic trials on the level of Siegert-networks.

The previous section did not include a quantitative analysis of simulated Siegert-RCNs versus their spiking equivalents. A more complete, comparative analysis of Siegert-RCNs and LIF-RCNs is currently in preparation and will hopefully be published soon. Our current experience in using Siegert-RCNs makes us believe that this approach is useful for certain kinds of investigations.

Besides the question of the circumstances under which a Siegert-network performs like its spiking equivalent, Siegert neurons are in any case a potent alternative to other rate-coded neuron models like linear threshold units with linear or sigmoidal activation functions [**WL90, Ros58, Abr02, MP43**].

The primary reason to prefer Siegert nodes to other rate-coded units is their similarity to LIF-neurons. Where other rate-coded neuron models come with an incompatible set of parameters as compared with LIF models, Siegert- and LIF-neurons have the same ones. This makes a transition between these two neuron models much faster and easier.

Chapters 9 and 10 will show two types of networks in which Siegert-RCNs are used as central building blocks. The proposed systems can perform interesting learning tasks, and the learned weights can then be used in neuro-computationally interesting ways, much like the network of Chapter 3, but more flexible.

Part 3

Learning in Recurrent Competitive Networks

CHAPTER 9

Topology Learning With Recurrent Competitive Networks

9.1. Introduction

In Chapter 5 we introduced RCNs and had a first look at their dynamics and response properties to population-coded inputs.

In this chapter we introduce a biologically plausible, Hebbian learning mechanism that alters the recurrent, excitatory RCN connections. In Section 5.3 we have already summarized the similarities and differences between RCNs and classical competition models (CCMs) like the winner-take-all (WTA) strategies used in Part 1 of this thesis. The learning procedure described here further increases the similarity between RCNs and CCMs.

The main result is that an RCN is capable of learning the input-topology. If the input is a one-dimensional population code, a trained RCN responds to complex stimuli very much like one-dimensional WTA networks do. On the other hand, if we feed a two-dimensional population code to the same network, the topology-learned connections causes this network to behave like a two-dimensional WTA.

An RCN's ability to learn the input topology (input encoding) is an important step in order to build systems that can adapt to complex inputs of a-priori unknown dimensionality or structure.

In contrast to that, systems that use CCMs can only work well if the person designing it has the right prior knowledge or makes some correct assumptions about the input the network will finally have to process.

9.2. Methods

9.2.1. Network Structure and Network Simulation.

Model Neurons. The model neuron we will use in this chapter is the Siegert node introduced in Chapter 8. We have seen that these nodes show input-response properties like leaky integrate-and-fire (LIF) neurons, but their big advantage is a significant computational speedup.

Another reason to use this neuron model is that the parameters of Siegert nodes and LIF neurons are identical. This offers the possibility of switching back and forth between a network of Siegert nodes or spiking LIF units.

Synaptic Connections. Synaptic connections between Siegert nodes are set to the biologically plausible parameter regime described in Chapter 6. (With the only that Siegert nodes cannot take slow $NMDA$ or $GABA_B$ currents into account.) The only free parameter per synapse is its weight, which denotes the total post-synaptic current (PSC) received at the soma of the receiving cell.

Network Structure. The RCN setup introduced in Chapter 5 is the basis for the simulations in this chapter. Figure 9.1 shows the entire network. It consists of an RCN-layer (A) and an input population (X). The sparse recurrent connections between the pyramidal nodes (W_{AA}) of A are also shown in the figure.

Table 9.1 gives the parameters needed to fully specify the recurrent network structure. In addition to this RCN we have a population of input neurons. All of these input neurons, lets call them c_i, are excitatory. Their output rate can be clamped to some fixed rate c_i.rate from outside the network.

TABLE 9.1. The set of parameters for an RCN setup containing 256 input-, 256 excitatory- and 64 inhibitory neurons.

RCN parameters							
n_i, n_p	n_b	p_{ip}	p_{ib}	p_{bp}	p_{pb}	p_{pp}	p_{bb}
		–	–	0.25	0.25	0.50	0.50
256	64	w_{ip}	w_{ib}	w_{bp}	w_{pb}	w_{pp}	w_{bb}
		1.25	3.00	−2.00	3.00	1.25	−2.00

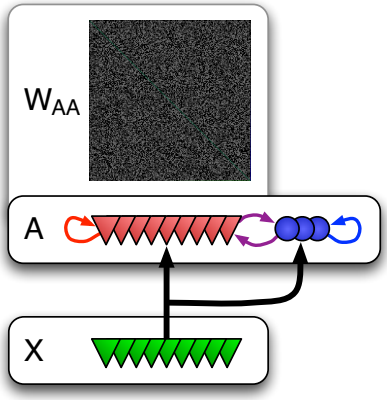


FIGURE 9.1. Network structure used for topology learning. It consists of an RCN-layer (A) and an input population (X). The sparse matrix (W_{AA}) shows the connection weights of the recurrent connections between the pyramidal cells of A.

The input population consists in total of n_p groups of n_i input neurons each. Each of the n_p pyramidal neurons in the RCN is assigned to exactly one of these groups, receiving input from exactly all these n_i nodes. A detailed explanation of the kind of input patterns we use in this chapter can be found in Section 9.2.3.

FFI Properties of this Setup. Figure 9.2 shows the FFI properties of the RCN setup given in Table 9.1. For a large range of different input strengths the output spike count stays basically the same. This is helpful for learning and using larger networks that contain RCNs, because (*i*) the learning dynamics can operate on a wide input range without the RCN's output activities degenerating, (*ii*) once the RCNs are trained their input will be complemented by increased lateral weights, so response stability during training requires robustness to input magnitudes, and (*iii*) trained RCNs need to be able to respond to very different input levels depending on how many neighboring RCNs are currently actively contributing input.

Homeostatic Activity Regulation (HAR). It is known that neurons actively regulate their spiking activity in order to maintain a certain activity level or activity range [**TN04, GA00, TN00**].

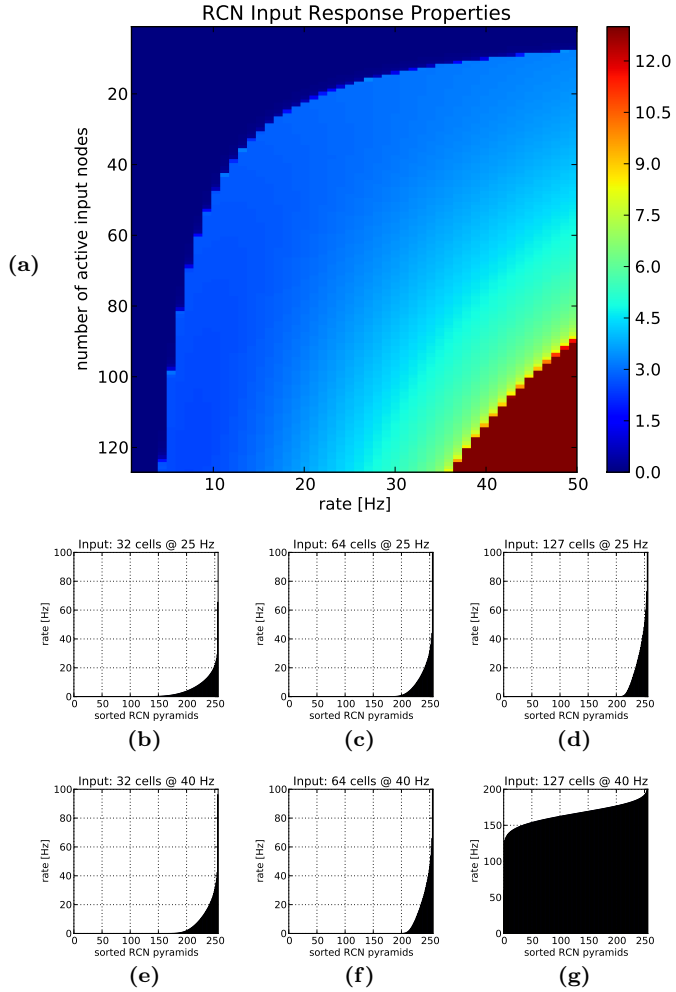


FIGURE 9.2. Demonstrates the capability of the network to generate sparse output patterns of similar total strengths for a fairly wide spectrum of input patterns. **(a)** The average output rate of all 256 pyramidal cells (color coded) when $y \in \{1..127\}$ of them receive an external input signal of $x \in \{1..50\}$ Hz. **(b-g)** Siegert-predictions (see text) of output distribution for different values of x and y. **(b)** $x = 25$ Hz, $y = 32$ cells receiving input; **(c)** $x = 25$ Hz, $y = 64$ cells receiving input; **(d)** $x = 25$ Hz, $y = 127$ cells receiving input; **(e)** $x = 40$ Hz, $y = 32$ cells receiving input; **(f)** $x = 40$ Hz, $y = 64$ cells receiving input; **(g)** $x = 40$ Hz, $y = 127$ cells receiving input.

In order to include HAR in our simulations we model a mechanism known as *Synaptic Scaling*. The basic idea is that a neuron which is too active (resp. not active enough), will scale down (resp. up) all its incoming synaptic connection weights by the same factor f_{HAR}. This causes the neuron's postsynaptic currents to be similarly scaled, thereby affecting its overall activity.

For convenience we actually do not change the individual synaptic weights in our simulations. Instead we give each neuron n a state variable n.har that is initialy set to be 1.0. Before using a synaptic weight w we will always compute the effective weight $w_{\text{eff}} = w \cdot n.\text{har}$, which is equivalent to the effect described above. (See e.g. line 20 in function **learnTopology**().) The Hebbian dynamics include weight normalization so the overall weight sum is thus fully controlled by the HAR dynamics.

Network Update Cycles. The Function **learnTopology**() shown below gives a detailed pseudo-code description of the activity updates. In line 28 we call the function **learnWeights**() that performs the Hebbian weight changes as described in Section 9.2.2.

Function **learnTopology**() provides an in-depth description of the training procedures we use. Here we provide some additional explanations:

Line 1-2**:** Set up the RCN network given in Table 9.1 and a population of $n_i \cdot n_p$ input neurons. Each group of n_i input nodes then gets connected to exactly 1 of the n_p pyramidal cells in the RCN. Each of the n_b basket cells receives connections from $\frac{n_b}{n_p} \cdot n_i$ uniformly sampled cells in each of these n_p input groups. This way of connecting the input population to the RCN follows Peters' rule [**BS91**].

Line 3-36**:** The main loop. This loop is run once per input pattern.

Line 5-8**:** A random input pattern is created and the input rates are assigned to the $n_i \cdot n_p$ input neurons. See Section 9.2.3 for detailed descriptions of used input patterns.

Line 10-35**:** The inner loop. Each cycle updates the entire network while keeping the same input pattern active. For all simulations performed in this chapter we used a value of $k_{\text{upd}} = 10$.

Line 12-17**:** Here we collect for each neuron a list of all rates and a list of corresponding weights of all inputs to this neuron, which encodes the current dendritic input. This is equivalent to the computation of the weighted sums of all inputs in classical rate-coded

Function **learnTopology** ():

create *inputPopulation* and *network*
connect *inputPopulation* to *network*

loop
  /* draw and feed input pattern */
  input = draw random input pattern
  for each n **in** *inputPopulation*.neurons **do**
    set n.rate to corresponding value in *input*
  end for

  /* per input we perform k_{upd} update steps */
  for $step = 1 \rightarrow k_{\text{upd}}$ **do**

    /* collect input to neuron n */
    for each n **in** *network*.neurons **do**
      for each (*sender*,*weight*) **in** $n.W_{\text{AA}}$ **do**
        n.inputRates.append(*sender*.rate)
        n.inputWeights.append(*weight*)
      end for
    end for

    /* compute firing rates */
    for each n **in** *network*.neurons **do**
      newRate = **Siegert**(n.inputRates, n.inputWeights $\cdot$ n.har)
      n.rate = $(1 - \tau_{\text{rate}}) \cdot n.\text{rate} + \tau_{\text{rate}} \cdot newRate$

      /* update har-value */
      if n is an excitatory neuron **then**
        n.har += $c_{\text{har}} \cdot (n.\text{desiredRate} - n.\text{rate})$
      end if
    end for

    /* call learning procedure */
    if $step \leq k_{\text{lrn}}$ **then**
      call **learnWeights**(*network*,W_{AA})
    end if

    /* reset values for next round of input */
    for each n **in** *network*.neurons **do**
      empty lists n.inputRates, n.inputWeights
    end for
  end for
end loop

network simulations. Note that Siegert nodes need more information than the weighted sum. (See Chapter 8 for details.)

Line 19-26**:** This code-block uses the dendritic input collected in lines 12-17 and the neurons internal homeostatic state to compute the next output rate of each neuron in the RCN.

Line 20**:** Here the function **Siegert**(...) is called in order to compute the output rate of an LIF neuron receiving the collected input rates with the corresponding synaptic weights. (See Chapter 8 for a detailed description of this function.) Note that the input weights are multiplied with the neuron's internal HAR-value. See Section 9.2.1 for further details on this homeostatic mechanism.

Line 21**:** Here, finally, we update the output rate of neuron n. The parameter τ_{rate} determines how quickly we approach $newRate$, the firing rate computed in line 20. In the simulations performed in this chapter we used the value $\tau_{\text{rate}} = 0.25$.

Line 24**:** Depending on how much the current activity is off the desired average activity level (n.desiredRate) we increment or decrement n.har. Recall that we used this variable to alter the input to function **Siegert**(...) in line 20. All simulation results for this chapter used the value $c_{\text{har}} = 0.0025$.

Line 28**:** Here, for the first $k_{\text{lrn}} \leq k_{\text{upd}}$ update steps after a new input is presented to the network and after all firing rates of all neurons in the network have been updated, we call function **learnWeights**(). This function modifies synaptic weights between individual pyramidal cells in our network. See Section 9.2.2 for further details. All simulation results in this chapter were created using the value $k_{\text{lrn}} = 0.3 \cdot k_{\text{upd}} = 3$. The motivation for k_{lrn} to exist in the first place is discussed in Section 9.2.2

Line 32-34**:** This code-block resets all temporary variables. This has to be done before the inner loop can be re-iterated.

9.2.2. The Learning Rule. All RCNs used in previous chapters where non-plastic, meaning that all connection weights where fixed throughout the duration of the entire simulation. Here we finally introduce plastic weights. More specifically, the recurrent weights among excitatory (pyramidal) cells is modified by a Hebbian learning rule.

The function **learnWeights** ($network$,W) describes the used learning rule by giving an intuitive pseudo-code language. The weight modification happens in line 6. The expressions c.pre and c.post denote the presynaptic and postsynaptic neuron, respectively.

Function **learnWeights** (*network*,W):

for each n **in** *network*.neurons **do**
  $oldSum = newSum = 0$
  /* compute new weights */
  for each c **in** $n.W$ **do**
    $oldSum \mathrel{+}= c.\text{weight}$
    $c.\text{weight} = c.\text{weight} + \alpha \cdot (c.\text{pre.rate} \cdot c.\text{post.rate})^k$
    $newSum \mathrel{+}= c.\text{weight}$
  end for
  /* normalize weights */
  for each c **in** $n.W$ **do**
    $c.\text{weight} = c.\text{weight} \cdot \frac{oldSum}{newSum}$
  end for
end for

This learning procedure does not have to be called after each update step in function **learnTopology** (). Only the first k_{lrn} of the k_{upd} steps are used for learning. The smaller the ratio $\frac{k_{\text{lrn}}}{k_{\text{upd}}}$, the more the final weight matrix will be influenced by the input itself. It turns out that in cases where the recurrent excitatory connections (W_{AA}, W_{BB}) are relatively strong, all learned weight matrices have the tendency to become block matrices, which leads to leads to slightly different dynamics from the ones we aim for in this chapter. More precisely, recurrent block matrices make the system settle in one of several discrete states. Although this can be a desirable behavior for some systems, here we aim for a continuous solution where the learned matrices are smooth.

Even from a biophysiological point of view it might be possible to influence the ratio $\frac{k_{\text{lrn}}}{k_{\text{upd}}}$. Experimental evidence suggests that backpropagating action-potentials (BAPs) are necessary for LTP [**FFdSCB10**]. The propagation of these BAPs are based on a chain reaction that is much less reliable than the one used for action potential propagations in axons. Inhibitory currents (GABA currents) in dendritic branches can disrupt this chain reaction [**WSS05**], which in turn renders the synapses along subsequent branches unable to perform further LTP. This effect grows with GABA-currents transmitted between the cells in a neural population. It is initially low but increases after stimulus onset.

With the constant k used in line 6, we can control whether the weight changes are linear ($k = 1$), superlinear ($k > 1$), or supralinear ($k < 1$) with the product of c.pre.rate and c.post.rate, the rates of the presynaptic and postsynaptic neuron, respectively. For $k = 1$ this rule is a classical Hebbian learning rule with learning rate α. In the simulations performed in this chapter we used the values $\alpha = 0.04$ and $k = 2$. Using $k = 1$ leads to very similar simulation results where the strong weights along the diagonal of Figure 9.4 would be a bit wider (data not shown).

Currently, the most prominent biological learning rule is spike time dependent plasticity (STDP) [**DP92, MLFS97, AN00**]. The reason to use a value $k > 1$ is motivated by neurophysiological findings (see e.g. [**ID03**]).

The loops in line 1 and 4 iterate over all synapses in the network. The third loop in lines 10-12 normalizes all incoming weights to a given neuron in order to maintain the same sum. This allows the HAR dynamics to have full control over the effective weight sum, as described on page 111.

9.2.3. Input Patterns. To demonstrate topology learning in RCNs we feed three kinds of inputs to three identical copies of the same network. For all results shown here neither the network nor any other simulation parameter was modified between these three cases. The only difference between the subsequently presented results is the input itself.

All inputs we use are wrapping around in order to avoid border effects. In other words, all input topologies we use are tori, where the smallest and largest values in each dimension coincide. For better readability, this wrap-around nature is omitted from the mathematical description given below.

After choosing which type of input to use, we start the function **learnTopology** (), which first sets up the network and connects $n_i \cdot n_p$ input neurons to the $n_p + n_b$ many neurons in the RCN (in lines 1-2). Each presented input is then drawn uniformly at random from the input class we choose (lines 5-8).

In this section we define these input classes and show how to draw random instances uniformly at random from those.

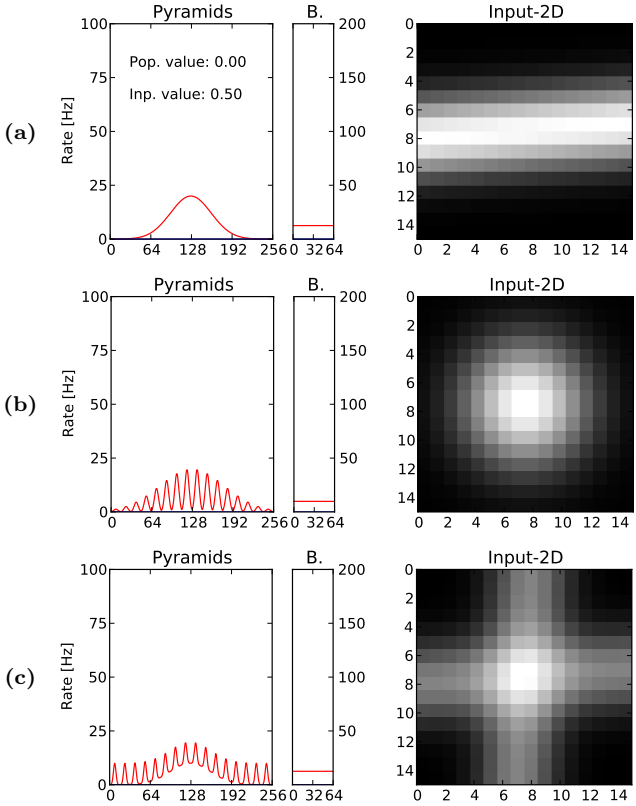


FIGURE 9.3. The three types of input used throughout this chapter. The left sides show the input rates fed from the $n_i = 256$ input neurons groups G_i to the pyramidal and basket cell population. The right sides show the same data by color-encoding the input rate (white being highest rate, black being zero). In addition we rearranged these 256 values in a 2-dimensional grid of size 16^2. **(a)** A one-dimensional (1D) population-coded input pattern centered at $x = 0.5$. **(b)** A two-dimensional (2D) population-coded input pattern centered at $\mathbf{v} = (0.5, 0.5)^T$. **(c)** Two one-dimensional (2x1D) population-coded input patterns, one at position $x = 0.5$, the other at position $y = 0.5$.

9.2.3.1. *One-Dimensional Input (1D).* This input type is similar to the kind of input we used in Part 1 of this thesis. The input population encodes a single scalar value $x \in [0, 1]$ using a population code.

There are n_p groups of input neurons. Each group G_g, $g \in \{1, \ldots, n_p\}$, consists of n_i individual cells. Every cell $c_{g,i} \in G_g$ has a preferred input stimulus value $x_{\text{pref}}(g) = \frac{g}{n_p}$ assigned to it.

We use a gaussian function $f_{\sigma,r_{\max}}(x)$ as the population activation pattern:

$$f_{\sigma,r_{\max}}(x) = r_{\max} \cdot e^{-(\frac{x^2}{2 \cdot \sigma^2})} \tag{9.1}$$

The peak of f is at $r_{\max}$, which corresponds to the maximum input frequency in Hz.

For input x, each input cell's output rate is then given by:

$$r(c_{g,i}, x) = f_{\sigma,r_{\max}}(x_{\text{pref}}(g) - x), \tag{9.2}$$

the gaussian function applied to the difference between x and the cells preferred value $x_{\text{pref}}(g)$. Figure 9.3(a) shows an example of a 1D-input for $x = 0.5$.

We choose a random input pattern by drawing a random value x uniformly at random from the unit interval. All results shown below are obtained using the parameters $\sigma = \frac{n_p}{8} = 32$ and $r_{\max} = 40$ Hz.

9.2.3.2. *Two-Dimensional Input (2D).* This type of input encodes two independent scalar values $x \in [0, 1]$ and $y \in [0, 1]$.

To each of the n_p input neuron groups G_g, $g \in \{1, \ldots, n_p\}$, n_p being a square number, we assign a vector:

$$\begin{aligned} \mathbf{v}_{\text{pref}}(g) &= \begin{pmatrix} x_{\text{pref}}(g) \\ y_{\text{pref}}(g) \end{pmatrix}, \text{with} \\ x_{\text{pref}}(g) &= \frac{g \mod \sqrt{n_p}}{\sqrt{n_p}}, \text{and} \\ y_{\text{pref}}(g) &= \frac{\lfloor g/\sqrt{n_p} \rfloor}{\sqrt{n_p}}. \end{aligned} \tag{9.3}$$

This assigns the n_p input groups to the points in a quadratic, regular grid, spanning the unit square.

Since we have a two-dimensional input space, we use a two-dimensional gaussian function:

$$f^2_{\sigma,r_{\max}}(\mathbf{v}) = r_{\max} \cdot e^{-(\frac{x^2+y^2}{2\cdot\sigma^2})}, \tag{9.4}$$

with $\mathbf{v} = (x, y)^T$ as population activation pattern.

Each input cell's output rate is then given by:

$$r(c_{g,i}, \mathbf{v}) = f^2_{\sigma,r_{\max}}(\mathbf{v}_{\text{pref},g} - \mathbf{v}) \tag{9.5}$$

the value of f^2, shifted by $\mathbf{v}$, at the cells preferred value $\mathbf{v}_{\text{pref}}(i)$. Figure 9.3(b) shows an example of a 2D-input for $\mathbf{v} = (0.5, 0.5)^T$.

Random input patterns are created by drawing a random vector $\mathbf{v}$ uniformly at random from the unit square. All results shown below are obtained using the parameters $\sigma = \frac{\sqrt{n_p}}{5} = 3.2$ and $r_{\max} = 40$ Hz.

9.2.3.3. *Two One-Dimensional Inputs (2x1D).* This type of input, like the previous one, encodes two independent scalar values $x \in [0, 1]$ and $y \in [0, 1]$.

To each of the n_p input neuron groups G_g, $g \in \{1, \ldots, n_p\}$, n_p being a square number, we assign a vector $\mathbf{v}_{\text{pref}}(g) = (x_{\text{pref}}(g), y_{\text{pref}}(g))^T$ as follows:

$$\begin{aligned} \mathbf{v}_{\text{pref}}(g) &= \begin{pmatrix} x_{\text{pref}}(g) \\ y_{\text{pref}}(g) \end{pmatrix}, \text{with} \\ x_{\text{pref}}(g) &= \frac{g}{n_p} \\ y_{\text{pref}}(g) &= \frac{(g \mod \sqrt{n_p}) \cdot \sqrt{n_p} + \lfloor g/\sqrt{n_p} \rfloor}{n_p}. \end{aligned} \tag{9.6}$$

Although we have a two-dimensional input-space, we use two one-dimensional gaussian functions $f_{\sigma,r_{\max}}(x)$ and $f_{\sigma,r_{\max}}(y)$ (see Equation 9.1) as population activation patterns.

Each input cell's output rate is then given by:

$$\begin{aligned} r(c_{i,j}, \mathbf{v}) &= f_{\sigma,r_{\max}/2}(x_{\text{pref}}(g) - x) + \\ &\quad f_{\sigma,r_{\max}/2}(y_{\text{pref}}(g) - y), \end{aligned} \tag{9.7}$$

the sum of the 2 gaussian functions, one fed to the unit square in y-direction at height x, the other fed to the unit square in x-direction at height y. Figure 9.3(c) shows an example of a 2x1D-input for $\mathbf{v} = (0.5, 0.5)^T$.

Random input patterns are created by drawing a random vector $\mathbf{v}$ uniformly at random from the unit square. All results shown below are obtained using the parameters $\sigma = \frac{\sqrt{n_p}}{8} = 32$ and $r_{\max} = 40$ Hz.

9.3. Results

In the previous sections we have set the stage for learning the input topology by modifying an RCN based only on the fed inputs. We described the simulation's update routine in the Function **learnTopology**() on page 112 and the learning dynamics in Function **learnWeights**() on page 114. The three types of input we use were described in Section 9.2.3.

Here we will first have a look at the weights resulting from the learning procedure. In a second step we show that the dynamics of such topology-learned RCNs are changing significantly, eventually getting very close to the dynamics of CCMs such as the WTA-networks used in Chapters 3 and 4.

9.3.1. Learned Input-Topologies. We create the exact same RCN for each type of input described in Section 9.2.3. After letting Function **learnTopology**() run for a couple of hundred iterations, the recurrent weights among the pyramidal cells converge to a quasi-stable state. We use the term *quasi-stable*, because each additional input presentation still slightly modifies the weights, which have therefore not mathematically converged to a stable value. However, doing several thousand additional iterations shows that this quasi-stable weight distribution does not significantly change any more.

Figure 9.4 shows the connection matrix containing the plastic weights w_{pp} between all pairs of pyramidal cells.

Although the weight matrix does, of course, contain all the data we are interested in, it is not so easy to interpret the results for 2-dimensional inputs (Figure 9.4(c,d)). Each line l of these matrices contains all outgoing weights coming from pyramidal cell c_l. A column m, in contrast, contains all incoming synapse weights cell number c_m receives from all its peers. For 2-dimensional inputs it is easier to interpret this data in a de-linearized visualization, where this kind of data is made consistent with the 2D-grid the input is based upon.

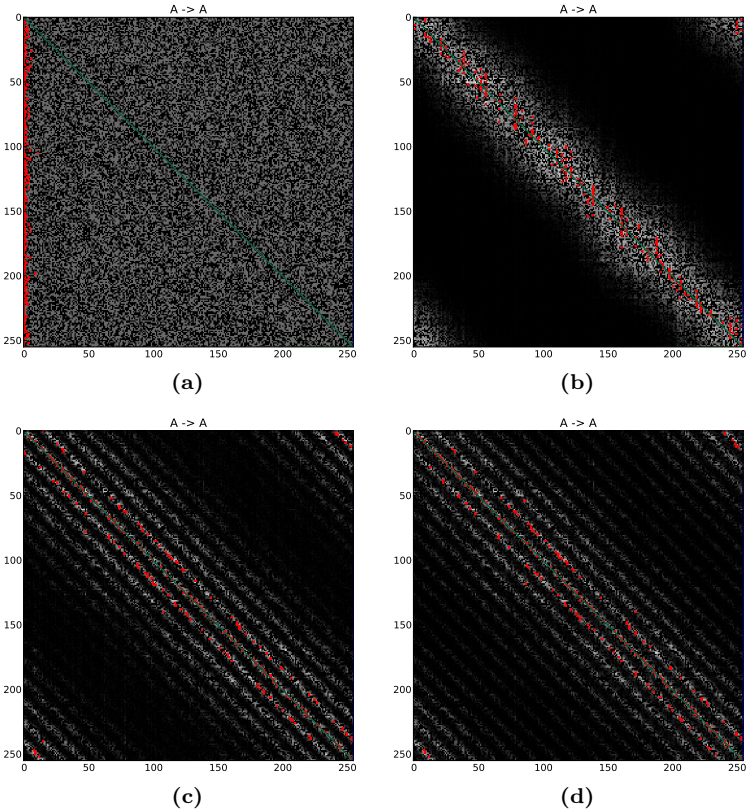


FIGURE 9.4. Learned topologies. Each plot shows the learned recurrent weights among the pyramidal cells (w_{pp}). The weights are color coded (white encoding the strongest weight and black encoding zero). Red markers indicate the largest w_{pp}-value in each matrix row. (If there are multiple maxima, the red dot marks the first one occurring in each row.) The faint green line marks the diagonal. **(a)** The initial weights (before topology-learning). All non-zero weights are equal. **(b)** Quasi stable weight matrix after some hundred random 1D-inputs have been presented to the network. **(c)** Quasi stable weight matrix after some hundred 2D-inputs have been presented to the network. **(c)** Quasi stable weight matrix after some hundred 2x1D-inputs have been presented to the network.

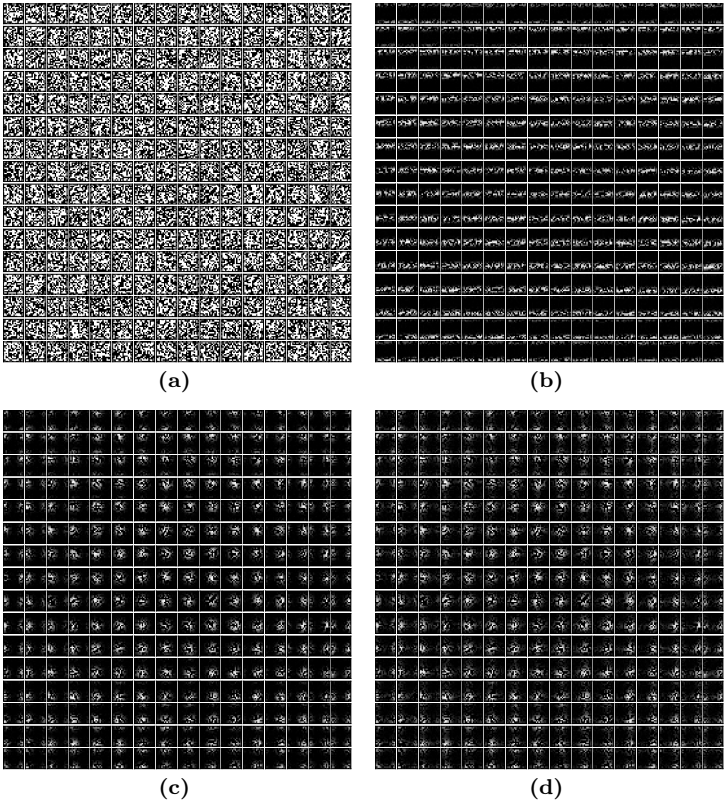


FIGURE 9.5. Learned topologies (weights grouped to 2D-tiles). Each plot shows the same w_{pp}-data as Figure 9.4(a-d). The weights shown in one tile correspond to the weights coming from one row in these weight matrices. This alternative visualization is mainly useful for two-dimensional topologies because it shows well how the strong inputs distribute over this 2D space (see text for details). **(a)** Initial weights. Weights are uniformly distributed across the entire population. The network itself does not offer any useful notion of proximity among its units. **(b)** Weights learned from 1D-inputs. Weights are tuned to only one of the two input dimensions. Note that the inputs to neurons tuned for a certain input value (vertical position of a tile in the 16×16 grid of tiles) are from other neurons that are tuned to similar values (vertical position of the strong weights inside the tile). **(c,d)** Weights learned by observing 2-dimensional input encodings (2D-inputs and 2x1D-inputs). Also here we can see that the strong inputs to a neuron (white clusters inside the tiles) come from its neighbors. That is, the relative position of the tile in the array of tiles coincides with the strongest weights inside the tile.

This kind of re-grouping of weights w_{pp} is shown in Figure 9.5. The data from each of the $n_p = 256$ rows from Figure 9.4 is rearranged (delinearized) onto a grid of size 16^2. We call one such small square a tile. The 256 resulting tiles we arrange another time on a grid of size 16^2. The tiles show in an intuitive way from where in the layer each node gets most of its input.

It can be seen that for all three types of input the strongest weights in the learned weight matrices are coming from cells with similar preferred input tuning (places that are close to each other with respect to the input topology).

It is important to step back and acknowledge the fact that the learned weights reflect the topology of the input, although the RCN initially only contains randomly drawn connections. Hence the network's initial connectivity and the topology of the input are very different in nature.

These results show that random networks, structurally similar to networks known to exist in local cortical structures (see Chapter 5), are capable of dealing with diverse input topologies of fundamentally different nature from the network's inherent "tabula rasa" topology [**KSM05**].

In the next section we will have a look at the dynamics of topology-learned RCNs when slightly more complex input patterns are being fed.

9.3.2. Using the Learned Topologies. Already in Chapter 5 we have seen that untrained RCNs containing only random connections share some features with CCMs. Section 5.3 contains a compact summary of similarities and differences between CCMs and RCNs.

Here we show how trained RCNs change their dynamics in comparison to untrained networks. All network effects we have mentioned in Chapter 5 are retained and emphasized. The main difference is that the inhibition in trained networks is operating on a pattern level and not only on the level of single cells. This means that competition between two simultaneously presented inputs, highly nonlinear winner selection, and partial pattern completion all become part of a trained RCN's network dynamics.

Figures 9.6-9.9 show the initial and settled network activities in an untrained as well as in a trained 1D-RCN when two inputs are fed simultaneously. The network fuses the inputs if they are similar enough (Figures 9.6 and 9.7). In cases where the given inputs are still similar enough to be fused, but one of the two inputs is weaker than the other,

the network strongly emphasizes this difference (Figure 9.8). These dynamics are very similar to the competition mechanisms proposed to exist in biological neural circuits [**DM07**] and also has striking similarities to the dynamics in CCMs [**Ama77, Hee92**].

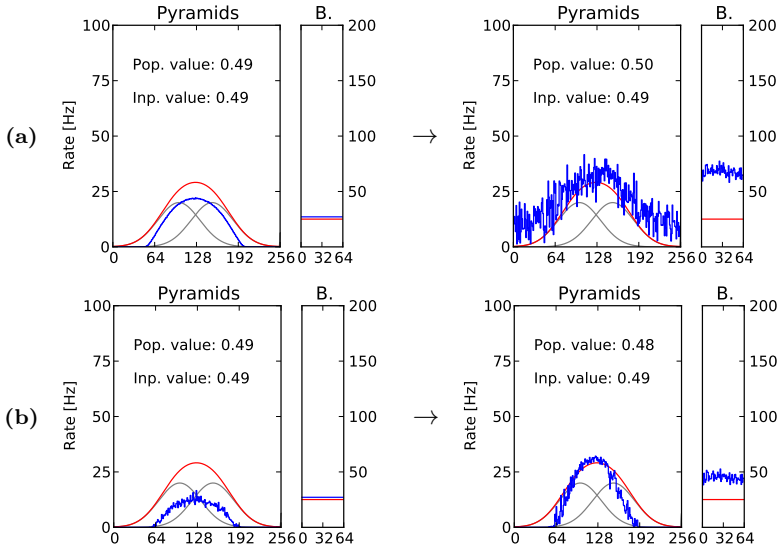


FIGURE 9.6. Initial responses (left) and settled responses (right) of two RCNs that were **(a)** untrained, and **(b)** trained using a sequence of randomly placed 1D-inputs. The population encoded values range from 0 (where the pyramidal cell with ID 0 is activated strongest) to 1 (where the pyramidal cell with ID 256 is activated strongest). Rows **(a,b)** are both generated by simultaneously feeding two inputs A (left grey curve) and B (right grey curve) to the network (total shown in red). In **(a)** the network responds by basically reproducing the shape of the input pattern (blue). In **(b)** we can see that the network activity is much sharper, almost as concentrated as the response to a single 1D-input would be.

Figures 9.10-9.13 show similar results to Figures 9.6-9.9, but in these cases the RCN was exposed to 2D inputs during the training phase of the network.

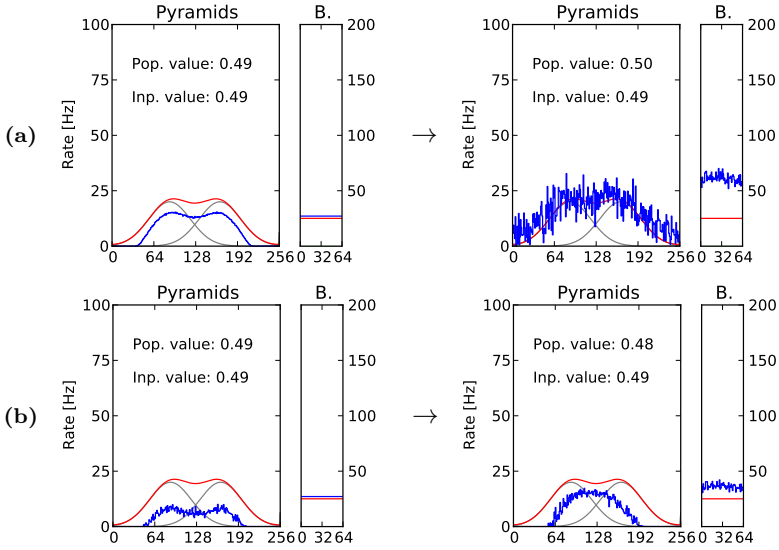


FIGURE 9.7. Initial responses (left) and settled responses (right) of two RCNs that were **(a)** untrained, and **(b)** trained using a sequence of randomly placed 1D-inputs. In contrast to Figure 9.6, inputs A (left grey curve) and B (right grey curve) encode more distant values, hence their peaks are further apart. In **(a)** the network responds by basically reproducing the bimodal input pattern. In **(b)** the network activity becomes unimodal, expressing a single peak of activity roughly at the center between the peaks of inputs A and B. The trained network has fused the two similar input patterns.

As already mentioned above, trained RCNs are capable of partial pattern completion. If a fed input pattern is incomplete, the lateral (recurrent) input from highly correlated pyramidal cells will help to restore the pattern. Figure 9.14 shows initial and settled network activities for partially omitted inputs for both, 1D and 2D.

9.3.3. Learning With Noisy Data. A striking feature of RCNs and the proposed training methods is the network's ability to cope with high levels of noise. The input patterns we fed up to now were all very clean

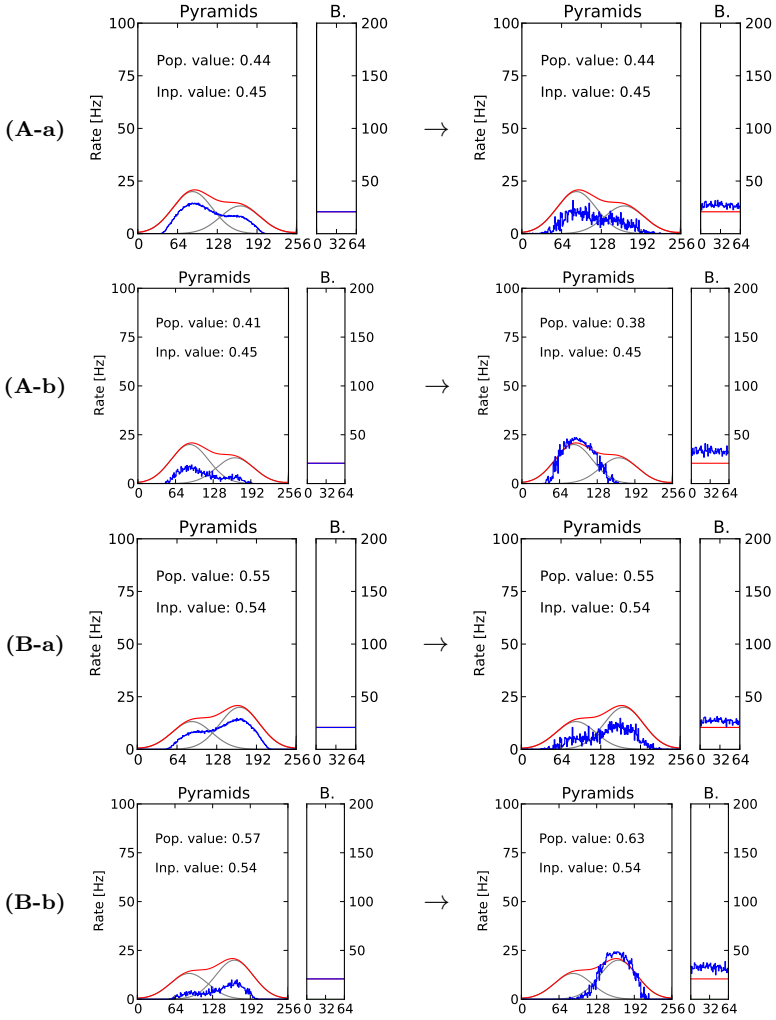


FIGURE 9.8. Initial responses (left) and settled responses (right) of two RCNs that were untrained **(A-a,B-a)**, and trained using a sequence of randomly placed 1D-inputs **(A-b,B-b)**. In contrast to Figure 9.7, inputs (left grey curve and right grey curve) do not have equal peak hight. Only the trained network in **(A-b)** and **(B-b)** inhibits the weaker input, picking the stronger input to win the competition.

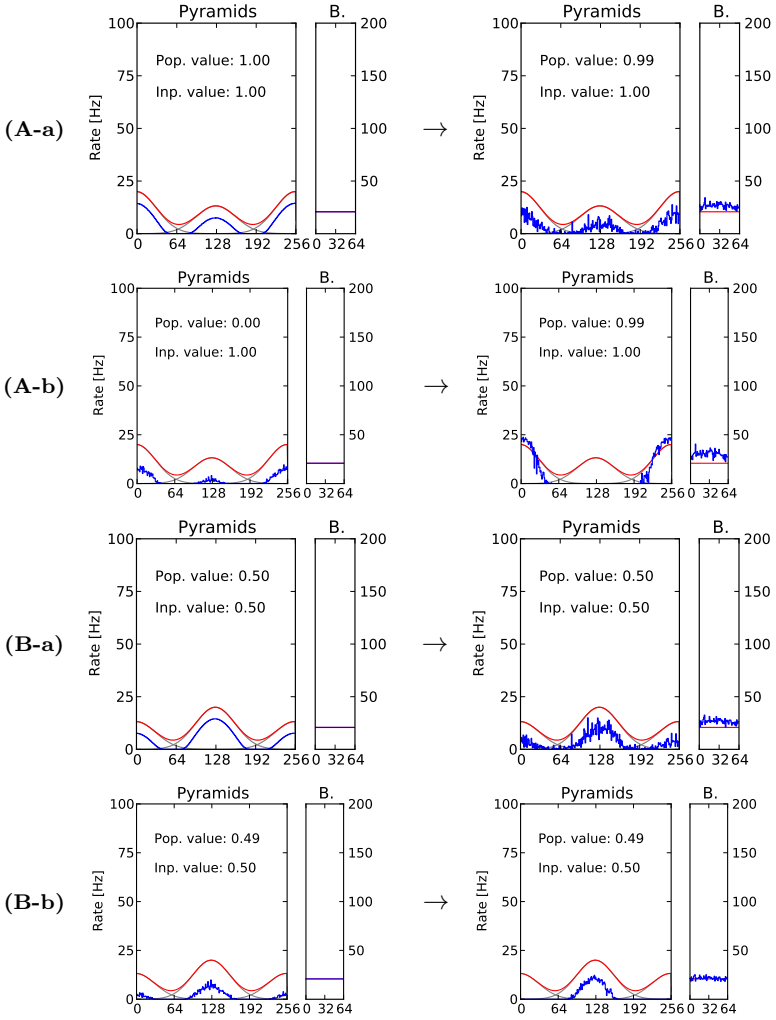


FIGURE 9.9. Initial responses (left) and settled responses (right) of two RCNs that were untrained **(A-a,B-a)**, and trained using a sequence of randomly placed 1D-inputs **(A-b,B-b)**. In contrast to Figure 9.8, the inputs (left grey curve and right grey curve, close to red sum) do encode two antipodal input values (as far apart as possible). As in Figure 9.8, only the trained networks in **(A-b)** and **(B-b)** inhibit the weaker input, picking the stronger input to win the competition.

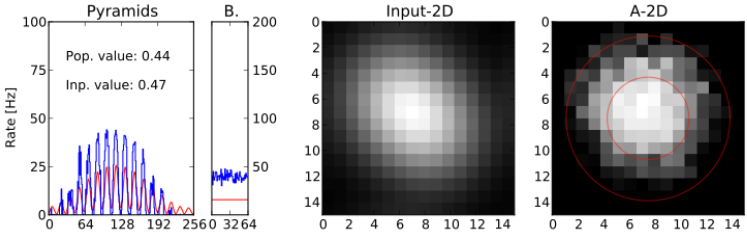


FIGURE 9.10. Input and settled response of an RCN that was trained using a sequence of randomly placed 2D-inputs. Here, two inputs encoding the values $\mathbf{v}_A = (0.4, 0.4)^T$ and $\mathbf{v}_B = (0.6, 0.6)^T$ are simultaneously fed. The network fuses the two inputs, treating them like one larger input encoding a value right between the individual inputs, roughly at $\mathbf{v}_{A+B} = (0.5, 0.5)^T$, shown in rightmost subfigure in red.

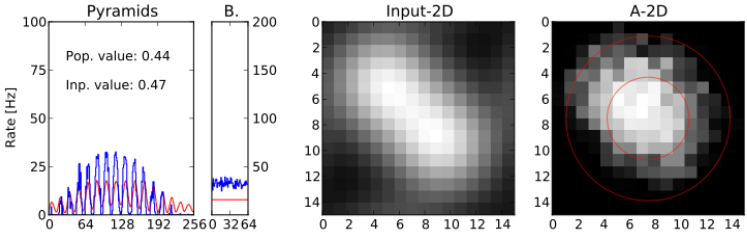


FIGURE 9.11. Input and settled response of an RCN that was trained using a sequence of randomly placed 2D-inputs. Here, two inputs encoding the values $\mathbf{v}_A = (0.35, 0.35)^T$ and $\mathbf{v}_B = (0.65, 0.65)^T$ are simultaneously fed. The network fuses the two inputs, treating them like one larger input encoding a value right between the individual inputs, roughly at $\mathbf{v}_{A+B} = (0.5, 0.5)^T$, shown in rightmost subfigure in red.

and smooth. In the following section we will show that this is absolutely not necessary. In fact we chose to show the results of experiments using noise free inputs mainly to give visually cleaner figures to the reader.

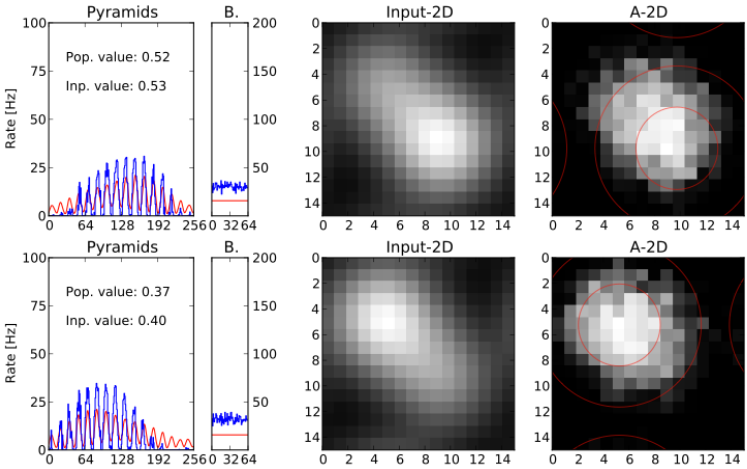


FIGURE 9.12. Input and settled response of an RCN that was trained using a sequence of randomly placed 2D-inputs. Here, two inputs encoding the values $\mathbf{v}_A = (0.35, 0.35)^T$ and $\mathbf{v}_B = (0.65, 0.65)^T$ are simultaneously fed. In contrast to Figure 9.11, the amplitudes of these inputs are set differently (the difference in amplitude height is 33%). Like in Figure 9.8 we see that the network non-linearly inhibits the weaker input. The red circles indicate the location of the stronger input.

9.3.3.1. *The Noise Model.* The network can cope with many types of noise. Here we show results for a relatively simple, gaussian noise model with uniform variance across the entire layer. After an input pattern is created as explained in Section 9.2.3, we use the peak input rate $r_{\max}$ and perturb each rate r_i to become

$$r_{i,\text{noisy}} = \max(0.0, \mathcal{N}(r_i, \frac{1}{4} r_{\max})) \tag{9.8}$$

The homeostasis (HAR) used in our model enables this method even to deal with some amount of non-symmetric, dependent noise. If, for example, each cell would be exposed to a noise source that does not depend on the current input but on the identity of this very cell, the input topology could still be learned (data not shown).

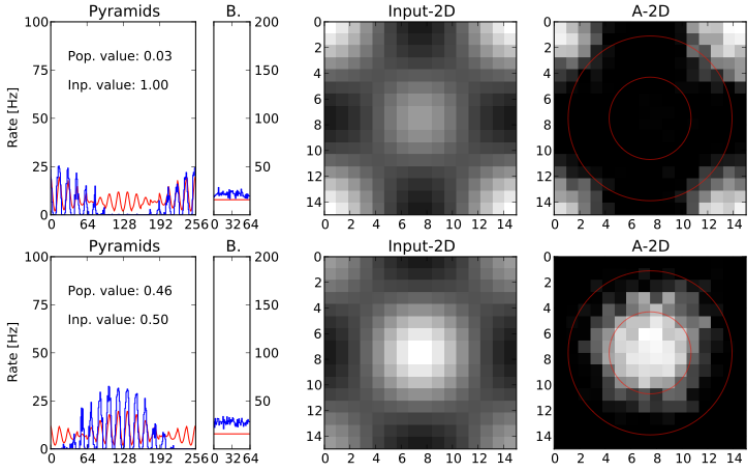


FIGURE 9.13. Input and settled response of an RCN that was trained using a sequence of randomly placed 2D-inputs. Here, two inputs encoding the values $\mathbf{v}_A = (0.0, 0.0)^T$ and $\mathbf{v}_B = (0.5, 0.5)^T$ are simultaneously fed. In contrast with Figure 9.12, these locations are chosen to be as far apart as possible. As in Figure 9.9, we see that the network nonlinearly inhibits the weaker input.

9.4. Discussion

When introducing RCNs in Chapter 5 we saw in Section 5.3.1 that these networks show dynamics that share some features with CCMs. Although the connections in RCNs are sparse and drawn uniformly at random, we saw that the the network's response to a fixed input pattern is stable and robust. We also saw that the strength of the output can be controlled by balancing excitation and inhibition.

Since we know that cortical synapses are plastic, it was a natural step to let the excitatory connections in RCNs be plastic as well. A combination of Hebbian learning and homeostatic activity regulation turned out to be suitable for learning the input topology, just by observing a stream of input patterns (Figures 9.4 and 9.5).

The main difference between trained and untrained networks is that recurrent excitatory inputs to a cell c_i are no longer sampled uniformly

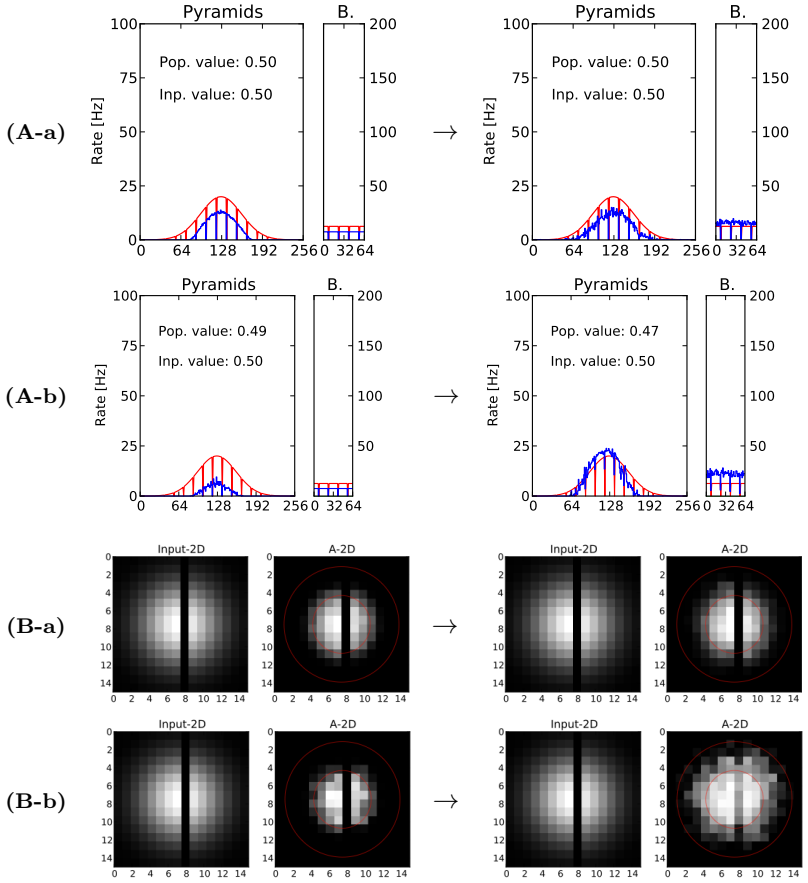


FIGURE 9.14. Signal restoration capabilities of trained and untrained RCNs. Initial responses (left) and settled responses (right) of two RCNs that were untrained in **(A-a,B-a)**, and trained with 1D-inputs in **(A-b)** and 2D-inputs in **(A-b)**. **(A-a,B-a)** show that the untrained networks do not fill in missing parts of the input. **(A-b)** Although some of the inputs are set to zero (ditches along red curves), the recurrent excitation within the trained RCN partially fills in these gaps. **(B-b)** Restoration of 2D-input. Although the input pattern is missing a vertical stripe, the settled network activity does not drop to zero at these locations as it did in the untrained network.

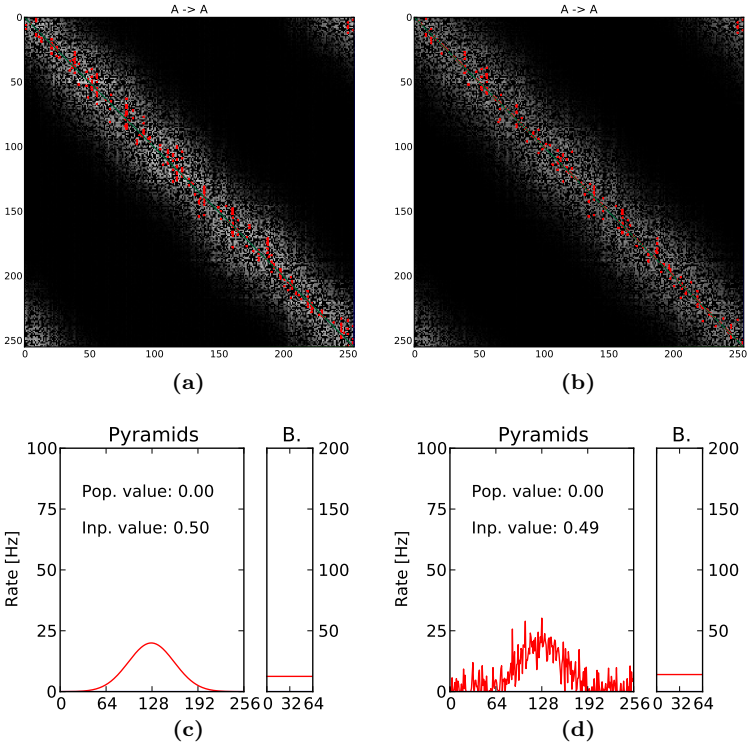


FIGURE 9.15. 1D topologies learned with continuous and noisy inputs. **(a,b)** Learned recurrent weights among the pyramidal cells (w_{pp}). The weights are color coded (white encoding the strongest weight and black encoding zero). Red markers indicate the largest w_{pp} value per matrix row. **(a)** Quasi stable weight matrix after some hundred *smooth* 1D-inputs have been presented to the network. (Same data as shown in Figure 9.4(b).) **(b)** Quasi stable weight matrix after some hundred *noisy* 1D-inputs have been presented to the network. The learned weight matrices turn out to be very similar although the inputs used to learn subfigure (b) were exposed to a high level of noise. **(c)** Example of a smooth, continuous input pattern. **(d)** Example of a noisy input pattern (see Equation 9.8).

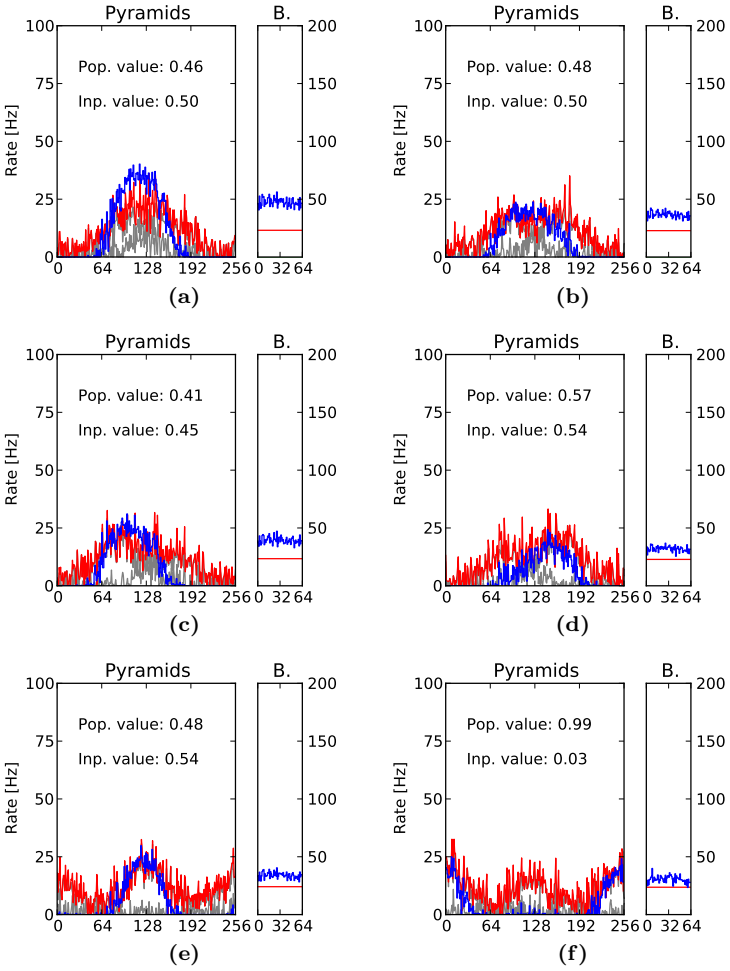


FIGURE 9.16. Settled responses of an RCN that was trained using a sequence of noisy 1D-inputs. **(a-f)** Noisy inputs (red curves) and corresponding outputs (blue curves). **(a)** Noisy equivalent to the data shown in Figure 9.6. **(b)** Noisy equivalent to the data shown in Figure 9.7. **(c)** Noisy equivalent to the data shown in Figure 9.8(A). **(d)** Noisy equivalent to the data shown in Figure 9.8(B). **(e)** Noisy equivalent to the data shown in Figure 9.9(A). **(f)** Noisy equivalent to the data shown in Figure 9.9(B).

from all its peers. The inputs are now clustered among fewer cells that define the learned neighborhood for c_i. This, in turn, defines a low-dimensional topology and a notion of expected population activity shape.

This shift from sparse, random connectivity to topology-based weights enables RCNs to introduce competition on the level of output patterns. Although competition (inhibition) was present also before the topology of the input was learned, it could only operate on the level of single cells. (This was visible in Figures 9.6-9.9(a)) The untrained network did not have any notion of frequently seen input or output patterns and therefore could not take them into account.

Since information about the type of input patterns and the corresponding network responses (output patterns) is encoded after training in the recurrent excitatory connections, activity patterns in the network start to compete on this higher semantical level. Not only do trained RCNs show a nonlinear winner selection similar to WTA networks [**DM07**], they are also capable of partially restoring fragmented input signals (see Figures 9.6-9.9 and Figure 9.14).

This makes RCNs, after learning the topology of the input, behave very much like CCMs [**Ama77, Hee92**].

9.4.1. A Probabilistic Interpretation of Trained RCNs. There are important differences between the dynamics of trained RCNs and the dynamics of most CCMs. In the following paragraphs we will see how an RCN's WTA dynamics can be interpreted as being probabilistic in nature.

A probabilistic interpretation of population-coded activity is getting more widely accepted [**MBLP06, BD11, DLP01, DLP99**]. In such a context, a population code is not assumed to only encode one of d scalar values but carries some kind of credibility or certainty value encoded in the way the activity is distributed over the population of neurons. If the population code was known the properties of a typical activity profile could be derived. However, we usually face the inverse problem of deriving the distribution from sample activity profiles.

9.4.1.1. *Normalization of the Output Activities.* Although a population code is not an arbitrary probability distribution but comes in general

with unknown constraints that can not easily be formalized[1], a population's total activity should not increase if it broadens its profile. Figures 5.4 and 9.2 show that RCNs come with exactly this output normalization feature when recurrent inhibition balances recurrent excitation.

In Figure 5.4 and Figure 9.2 we have seen that the total input strength to an RCN, once it passes some minimal input strength needed to excite the receiving population, can be increased multiple times without changing the RCNs output strength and output distributions all that much. Of course this is not the same as a normalization in a mathematical sense, but it might very well suffice for some kinds of neural computations [**CH11**].

9.4.1.2. *How Input and Output Activities Relate to Each Other.* In Figures 9.6-9.9 we saw that the activity patterns in untrained RCNs follow the given input distributions very well. This was not the case for trained RCNs, where the output lump could be sharpened, widened or shifted to other peak positions.

In Bayesian terms this is because the training samples introduced a prior to the RCN that is influencing the relaxation of the network to a population-coded output.

This prior could be formalized as a learned probability distribution on the set of all outputs. Then a Bayesian approach might define the desired output $\sigma = (\sigma_1, \sigma_2, \cdots, \sigma_{n_e})$, n_e being the number of excitatory neurons in the RCN, as the unit vector that maximizes the joint probability $\Pr(\sigma, \alpha)$, where $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_{n_i})$ is the activity of the n_i input neurons.

The latter expression, $\Pr(\sigma, \alpha)$, can be computed as:

$$\Pr(\sigma, \alpha) = \Pr(\alpha \mid \sigma) \cdot \Pr(\sigma). \tag{9.9}$$

The probabilities on the right hand side can be modeled in many ways. For our approach we assume conditional independence of the input activity α, given by

$$\Pr(\alpha \mid \sigma) \quad = \quad \prod_i \Pr(\alpha_i \mid \sigma). \tag{9.10}$$

[1]One attempt to formalize probabilistic population codes can be found in [**MBLP06**]

We model input activities to be less likely when deviating away from the corresponding output activity, as

$$\Pr(\alpha_i \mid \sigma) \quad = \quad c_1 \cdot \exp(-|\alpha_i - \sigma_i|^{k_1}), \tag{9.11}$$

with k_1 being a constant determining the used distance measure and c_1 being a normalization constant.

The prior on the output activity, influenced by the learned correlation between α_i and α_j, $\mathcal{C}_{i,j}$, is given by

$$\Pr(\sigma) \quad = \quad c_2 \cdot \exp\left(-\kappa \sum_{i,j} |\mathcal{C}_{i,j} - f(\sigma_i, \sigma_j)|^{k_2}\right), \tag{9.12}$$

where κ determines the influence of the prior, k_2 is a constant defining the used distance measure, and c_2 is a normalizing constant. The prior ensures that pairs of output activities (σ_i, σ_j), with high learned correlations $\mathcal{C}_{i,j}$, have similar values.

One possible formalization for f, the similarity measure for output activities, is

$$f(x, y) = 1 - |x - y|. \tag{9.13}$$

Here we assumed that desired outputs (σ_i) and the activity of input neurons (α_j) are given as values $\in [0, 1]$.

Eventually we would like to maximize the left hand side of Equation 9.9 with respect to σ:

$$\sigma^* = \underset{\sigma}{\operatorname{argmax}} \Pr(\sigma, \alpha) = \underset{\sigma}{\operatorname{argmin}} \left(\underbrace{-\ln \Pr(\sigma, \alpha)}_{\mathcal{E}(\sigma,\alpha)}\right). \tag{9.14}$$

with the error function $\mathcal{E}(\sigma, \alpha)$ being

$$\mathcal{E}(\sigma, \alpha) = \sum_i |\alpha_i - \sigma_i|^{k_1} + \kappa \cdot \sum_{i,j} |\mathcal{C}_{i,j} - f(\sigma_i, \sigma_j)|^{k_2}. \tag{9.15}$$

The first term punishes outputs that deviate too strongly from the input, while the second term punishes outputs that are not compatible with the prior knowledge.

Hence, $\mathcal{E}$ formalizes a tradeoff between the input pattern α and the system's learned knowledge $\mathcal{C}$, previously called the prior. The dynamics of a trained RCN can now be understood as finding an energy minimum given in Equation 9.14.

Of course, the output generated by an RCN is not rigorously speaking the precise solution of such a minimization problem (or distributed according to Equation 9.9). However, qualitatively an RCN appears to compute a solution along the lines we have outlined here.

We have already mentioned before that the nonlinear effects (WTA dynamics) of trained RCNs can be controlled by the strength of the recurrent weights (w_{pp}). In light of the above mentioned probabilistic interpretation this is equivalent to being able to adjust the relative influence of the learned prior knowledge with the currently seen input distribution (κ).

Figure 9.16(a-f) provides a nice overview of single RCN dynamics under a series of different input patterns (mind that already the input strengths for different sub-figures are not strictly normalized).

Like before we can immediately see that the probabilistic computations are most likely not precise in a mathematical sense, but the observable dynamics are going in a good direction. We think that the remaining error is tolerable and large scale systems of coupled RCNs may be capable of approximating probabilistic computations.

In the next chapter we will first discuss how two trainable RCNs can be coupled in a way that can still learn the topology of the individual input patterns, while also learning the relation between these inputs. At the end of the next chapter we will continue the probabilistic interpretation we have started here.

CHAPTER 10

Relation Learning With Rate Coded Recurrent Competitive Networks

10.1. Introduction

In Chapter 9 we saw how an untrained, tabula rasa RCN can learn the topology of population encoded inputs. By learning the input topology the network obtained a notion of what an input pattern usually looks like, which enabled the trained RCN to let superimposed input values compete with each other.

In this chapter we are going to extend the network of Chapter 9. We will study the learning and evaluation dynamics of two coupled RCN layers. The coupling, as shown in Figure 10.1, is realized using sparse, random, excitatory connections.

These connections are, like the recurrent connections in each individual RCN layer, plastic. The weights of these connections are trained using exactly the same Hebbian learning procedure used in Chapter 9.

10.1.1. The Learning Task in a Network of Coupled RCNs. The network has to simultaneously accomplish two tasks: the first one is the adaption to the input encoding. This is, the network should maintain the topology learning capabilities of the individual RCN layers, which we studied in Chapter 9. Additionally we want the two layers to use their coupling to exchange information about the input each of them receives. The network should figure out how the population encoded input values relate to each other, and encode this relation in the afferent (coupling) weight matrices.

The second learning task is similar to the unsupervised learning of relations discussed in Chapter 3. Differences include: (*i*) the coupled

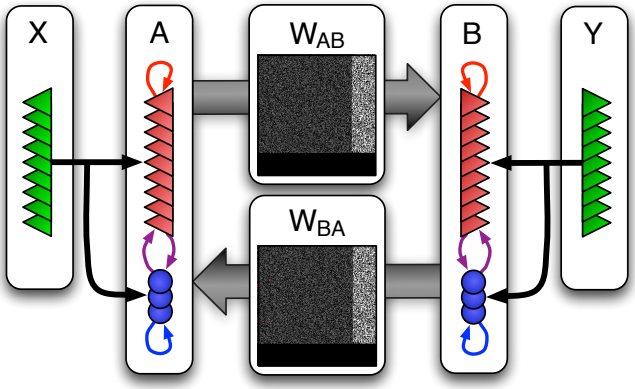


FIGURE 10.1. Network structure used for learning a relation. Populations X and Y are used to feed input to the network which consists of two layers A and B. A and B are each Siegert-RCNs set up in exactly the same way as the Siegert-RCN used in Chapter 9 to learn the topology of the input space. The activities of the pyramidal subpopulation of A are sent to layer B and vice versa, through weight matrices W_{AB} and W_{BA}. Both, the recurrent and inter-layer connectivities are sparse with only a fraction p_{pp} (resp. p_{pb}) of all possible connections present. All existing connections are initialized with the same weight w_{pp} for connections from pyramids to pyramids, and w_{pb} for connections from pyramids to basket cells.

RCNs are able to adapt to the input encoding received from the environment of the system, (*ii*) we use RCNs rather than CCMs for getting the desired WTA network dynamics, thus improving the neuroanatomical plausibility of the system, and (*iii*) the neuron model we use is not an artificial sigmoidal threshold unit, but a useful abstraction of leaky integrate-and-fire neurons (the Siegert neuron described in Chapter 8), which will eventually help to ease the transition back to a spiking neural network.

We believe that these differences, besides being interesting by themselves, will proove to be crucial modifications for building large-scale models that solve complex learning tasks.

10.1.2. Related Work. In Section 5.1.1 we placed our work on RCNs in relation to existing work. Since RCNs are the central module in the type of networks we are presenting here, the citations given in Section 5.1.1 also apply here.

In Chapter 9 we showed that trained RCNs and CCMs show very similar dynamics when fed with population-coded inputs. In this chapter we build larger networks that consist of such WTA-like modules. The same is true for the work by Rutishauser et al. [**RDS10, RD09**]. They show that CCM-like building blocks made of spiking units can be used to build arbitrary finite state machines. Their system follows quite different computational strategies, but is an interesting and very reliable way to build large and robust system that consist of modules having WTA dynamics.

Yet another, more theoretical approach to building larger systems out of WTA modules can be found in [**Maa00**]. However, we can not directly apply the results by Maass to our own systems because his understanding of a soft winner-take-all module differs significantly from ours.

Besides the work mentioned in the last paragraphs, the pointers given in Chapter 2 also contribute to a more complete picture of the scientific context of our work.

10.2. Methods

10.2.1. Network Structure and Network Simulation.

Model Neurons. The model neurons we use in this chapter are again the Siegert nodes introduced in Chapter 8. We have seen that these nodes show input-response properties like leaky integrate-and-fire (LIF) neurons while providing a significant computational speedup.

A reason to use this neuron model instead of other rate-coded neurons is that the parameters of Siegert neurons and LIF neurons coincide. This offers the possibility to switch back and forth between a network of Siegert nodes and spiking LIF units relatively easily.

Synaptic Connections. Synaptic connections between Siegert neurons are set to the biologically plausible parameter regime described in Chapter 6. (With the caveat that Siegert nodes cannot take slow $NMDA$ or $GABA_B$ currents into account.) The only free parameter per synapse is its weight, which denotes the total post-synaptic current (PSC) received at the soma of the receiving cell.

Network Structure. The RCN setup introduced in Chapter 5 is the basis for all simulations in this chapter. Figure 10.1 shows how the network is structured. It consists of two coupled RCN-layers (A and B) and two input populations (X and Y). The sparse recurrent connections between the pyramidal nodes within A (W_{AA}) and B (W_{BB}) are plastic[1], just as they were in Chapter 9. They allow the network to dynamically adapt to the topology of the fed inputs.

Layers A and B are coupled by the sparse random connections stored in W_{AB} and W_{BA}. These newly introduced connections will be trained using the same learning procedure as the one introduced in Chapter 9. Since these connections are not recurrent, but mediate the activities between layers, they learn the correlation between the activities in A and B. We will refer to the correlation in the data simply as the "relation" or "hidden relation".

Table 10.1 contains all parameters needed to fully specify the network. All input neurons c_i in X and Y are excitatory. Their output rate can be clamped to any fixed poisson rate 'c_i.rate' from outside the network.

The input population consists of n_p groups of n_i input neurons each. Each of the n_p pyramidal neurons in the RCN is assigned to exactly one of these groups, receiving input exactly from these n_i neurons. A detailed explanation of the kind of input patterns we use can be found in Section 9.2.3 in the previous chapter.

Note that the parameters we use here are the same as the ones we used to learn the topology within a single layer in Chapter 9 (see Table 9.1). The only difference is that the total strength of the recurrent connections is now divided in two halves. One is still learning the input topology, while the other half is now concerned with the relation between layers A and B.

[1]The weight matrices W_{AA} and W_{BB} are only symbolized as red arrows in Figure 10.1. They are like the weight matrix shown in Figure 9.1, fulfilling the same purpose as in Chapter 9.

TABLE 10.1. The set of parameters for a coupled RCN setup containing 256 excitatory and 64 inhibitory neurons per layer. All parameter names are as in Table 9.1. The new parameters p_{po}, p_{bo}, w_{po}, and w_{bo} ('o' as in output) are describing the weight of all connections in W_{AB} and W_{BA}.

Coupled RCNs: all parameters									
n_i,n_p	n_b	p_{ip}	p_{ib}	p_{po}	p_{bo}	p_{bp}	p_{pb}	p_{pp}	p_{bb}
		–	–	0.50	0.50	0.25	0.25	0.50	0.50
256	64	w_{ip}	w_{ib}	w_{po}	w_{bo}	w_{bp}	w_{pb}	w_{pp}	w_{bb}
		1.25	3.00	$\frac{1.25}{2}$	$\frac{3.00}{2}$	-2.00	$\frac{3.00}{2}$	$\frac{1.25}{2}$	-2.00

FFI Properties of this Setup. Since we are using the same RCN setup as in Chapter 9, Figure 9.2 also applies to the setup given in Table 10.1. For a large range of different input strengths the output spike count stays basically the same. This is helpful for learning and using larger networks that contain RCNs, because (*i*) the learning dynamics can operate on a wide input range without the RCN's output activities degenerating, (*ii*) once the RCNs are trained their input will be complemented by increased lateral weights, so response stability during training requires robustness to input magnitudes, and (*iii*) trained RCNs need to be able to respond to very different input levels depending on how many neighboring RCNs are currently actively contributing input.

Homeostatic Activity Regulation (HAR). It is known that neurons actively regulate their spiking activity in order to maintain a certain activity level or activity range [**TN04, GA00, TN00**].

In order to include HAR in our simulations we model a mechanism known as *Synaptic Scaling*. The basic idea is that a neuron which is too active (resp. not active enough), will scale down (resp. up) all its incoming synaptic connection weights by the same factor f_{HAR}. This causes the neuron's postsynaptic currents to be similarly scaled, thereby affecting its overall activity.

For convenience we actually do not change the individual synaptic weights in our simulations. Instead we give each neuron n a state variable n.har that is initialy set to be 1.0. Before using a synaptic weight w we will always compute the effective weight $w_{\text{eff}} = w \cdot n.\text{har}$, which is equivalent to the effect described above. (See e.g. line 20

in function **learnTopology**().) The Hebbian dynamics include weight normalization so the overall weight sum is thus fully controlled by the HAR dynamics.

Network Update Cycles. Function **learnRelation**() shown below gives a detailed pseudo-code description of the activity updates. In lines 29-33 we call the function **learnWeights**() that performs the Hebbian weight changes as described in Section 9.2.2 of the previous chapter.

Function **learnRelation**() provides an in-depth description of the training procedures that we use to learn the relation between two inputs of unknown topology. The algorithm is basically the same as the one on page 111, which we have explained in great detail there.

The biggest differences are (*i*) the input given to the network (see next section) and (*ii*) lines 29-33, where the learning procedure is called separately for each of the connection matrices W_{AA}, W_{BB}, W_{AB}, and W_{BA}.

10.2.2. The Learning Rule. The function **learnWeights** (...) described in the previous chapter in Section 9.2.2 is used to learn the local recurrent connections (W_{AA}, W_{BB}) as well as the efferent connections (W_{AB}, and W_{BA}) between layers A and B.

This learning procedure does not have to be called after each update step in function **learnTopology** (). Only the first k_{lrn} of the k_{upd} steps are used for learning. The smaller the ratio $\frac{k_{\mathrm{lrn}}}{k_{\mathrm{upd}}}$, the more the final weight matrix will be influenced by the input itself. It turns out that in cases where the recurrent excitatory connections (W_{AA}, W_{BB}) are relatively strong, all learned weight matrices have the tendency to become block matrices, which leads to leads to slightly different dynamics from the ones we aim for in this chapter. More precisely, recurrent block matrices make the system settle in one of several discrete states. Although this can be a desirable behavior for some systems, here we aim for a continuous solution where the learned matrices are smooth.

Even from a biophysiological point of view it might be possible to influence the ratio $\frac{k_{\mathrm{lrn}}}{k_{\mathrm{upd}}}$. Experimental evidence suggests that backpropagating action-potentials (BAPs) are necessary for LTP [**FFdSCB10**]. The propagation of these BAPs are based on a chain reaction that is much less reliable than the one used for action potential propagations in axons. Inhibitory currents (GABA currents) in dendritic branches can disrupt this chain reaction [**WSS05**], which in turn renders the synapses along subsequent branches unable to perform further LTP. This effect

Function **learnRelation** ():

create X, Y and A,B
connect $X \to$ A, $Y \to$ B, A $\to$ B, B $\to$ A

loop
  /* draw and feed input pattern */
  inputs = draw random input pattern for X and Y
  for each n_X **in** X.neurons **and** n_Y **in** Y.neurons **do**
    set n_X.rate to corresponding value in *inputs*.X
    set n_Y.rate to corresponding value in *inputs*.Y
  end for

  /* per input we perform k_{upd} update steps */
  for *step* $= 1 \to k_{\text{upd}}$ **do**

    /* collect input to neuron n */
    for each n **in** (A.neurons+B.neurons) **do**
      for each (*sender*,*weight*) **in** n.*inConns* **do**
        n.inputRates.append(*sender*.rate)
        n.inputWeights.append(*weight*)
      end for
    end for

    /* compute firing rates */
    for each n **in** (A.neurons+B.neurons) **do**
      newRate = **Siegert**(n.inputRates, n.inputWeights $\cdot$ n.har)
      n.rate $= (1 - \tau_{\text{rate}}) \cdot n.\text{rate} + \tau_{\text{rate}} \cdot \textit{newRate}$

      /* update har-value */
      if n is an excitatory neuron **then**
        n.har += $c_{\text{har}} \cdot (n.\text{desiredRate} - n.\text{rate})$
      end if
    end for

    /* call learning procedure */
    if *step* $\leq k_{\text{lrn}}$ **then**
      call **learnWeights**(*network*,W_{AA})
      call **learnWeights**(*network*,W_{BB})
      call **learnWeights**(*network*,W_{AB})
      call **learnWeights**(*network*,W_{BA})
    end if

    /* reset values for next round of input */
    for each n **in** (A.neurons+B.neurons) **do**
      empty lists n.inputRates, n.inputWeights
    end for
  end for
end loop

grows with GABA-currents transmitted between the cells in a neural population. It is initially low but increases after stimulus onset.

10.2.3. Input Patterns. All inputs to the individual RCN layers are of the 1D and 2D types introduced in Section 9.2.3 on page 117.

10.2.3.1. *The Relations Used to Test our Method.* We use two functional relations to test the proposed network. We will interpret all variable values for both relations to be in the range $[0, 1]$. Note that this is really only a matter of interpretation since network activities and the position of population encoded activity lumps are only meaningful through their relation to other populations.

The first relation we use is $y = x^2$, for which we need one-dimensional population-coded values in X and Y.

The second relation we will test the network with is formally described by $(y_1, y_2) = (1 - x_1, 1 - x_2)$ and will use two-dimensional population-coded inputs in X and Y. Since this relation takes a coordinate in the unit square and projects it across $(0.5, 0.5)$ to the opposite side we call this relation the "inversion relation".

When the function **learnRelation** () is called it instantiates the network and starts feeding data drawn uniformly at random from one of the mentioned example relations. (A description of the detailed input structure can be found in Section 9.2.3.) Note that neither network creation nor the choice of any parameter depends on the relation the network is finally exposed to. The strength of this network is its flexibility and robustness.

10.3. Results

Let's have a look how the proposed learning method affects (*i*) the recurrent weight matrices W_{AA} and W_{BB}, which are supposed to learn the topology of the input space as they did in Chapter 9, and (*ii*) the afferent weight matrices W_{AB} and W_{BA} that connect layers A and B with each other and are supposed to capture the hidden relation.

We will see that the network is capable of learning relations using one- as well as two-dimensional population codes. We want to stress once again that the network does not have to be altered in any way when changing the relation or population code structure of the input.

After training the network we will see that the learned connection weights enable the network to perform inference tasks, biased decision making, cue integration, and even solving constraint satisfaction problems.

All experiments of this chapter were repeated after adding a significant amount of noise to the training data. The results show that learning as well as the successive computational tasks are robust to noise.

The flexibility to adapt to different input topologies (input encodings), the robustness to noise, and the fact that coupled RCNs only consist of locally operating, biological concepts makes this kind of network an interesting model for cortical computation.

10.3.1. Learning a Nonlinear Relation. First we let the network learn a quadratic relation and see how the plastic random connections change. After that we will show some computationally interesting ways that the learned network can be used. At the end of this section we will elaborate on some difficulties in learning nonlinear relations.

We will train the network with the functional relation: $y = x^2$, where x and y are both single scalar values in $[0, 1]$. The activities in both input layers (X and Y) are encoded using one-dimensional population codes. A value fed to the network is encoded by an activity lump in X and Y, centered at position x and y, respectively. To make this clear let us consider an example. In case $x = 0.5$, the lump of activity in layer X will be at the center of the layer X (with respect to the ordering of the neurons in our figures, see also Section 9.2.3 on page 117). According to the relation $y = x^2$ we know that $y = 0.25$, meaning that Y's activity profile will peak around the cell with ID 64 (since there are 256 excitatory cells in total and $256 \cdot 0.25 = 64$).

Already after training on a few hundred examples that are drawn uniformly at random from all pairs (v, v^2) ($v \in [0, 1[$), the weight matrices W_{AA}, W_{BB}, W_{AB}, and W_{BA} are as shown in Figure 10.2.

In Figure 10.2 we can see that the recurrent matrices W_{AA} and W_{BB} did successfully learn the one-dimensional input topology. Further we see that the weights in W_{AB} and W_{BA} are describing the quadratic relation in the training data fairly well.

10.3.1.1. *How to Make Use of the Learned Relation.* It is natural to ask how the learned weights can be utilized after being trained.

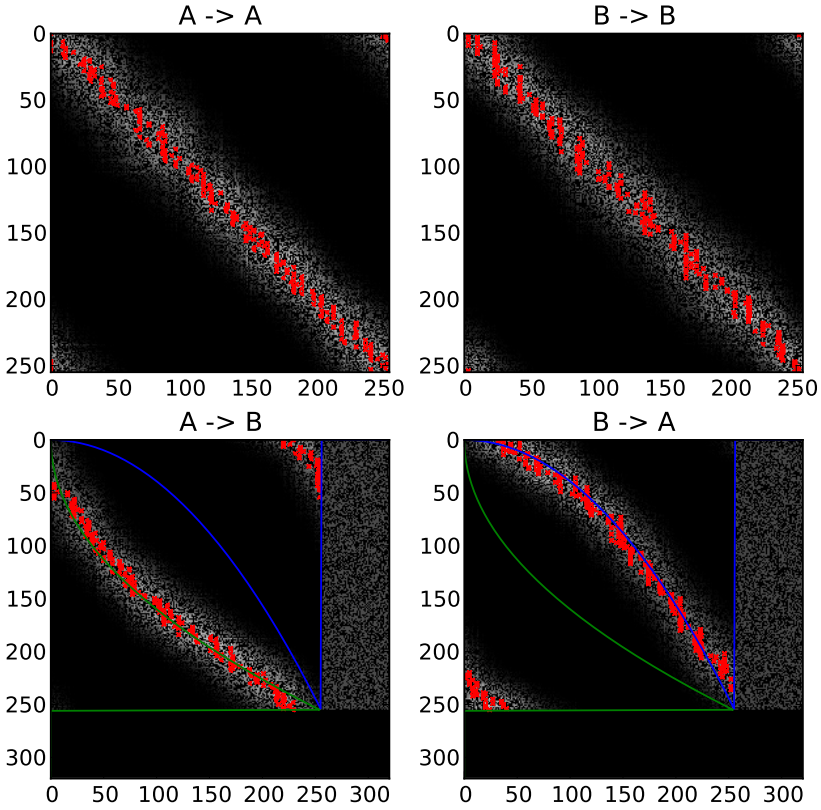


FIGURE 10.2. Learned lateral (topology) connections and afferent (relation) connections. The weights are color coded with white encoding the strongest weight and black encoding zero. Red dots indicate the largest weight in the corresponding matrix row. (If there are multiple maxima, the red dot marks the first one occurring in each row.) The plotted weights show the results of learning a quadratic relation. See text for further details. **(A→A)** The weights that learned the topology of the input X provided to layer A. **(B→B)** The weights that learned the topology of the input Y provided to layer A. **(A→B)** The weights from A to B. For reference, the faint green line shows the $y = x^2$ relation. **(B→A)** The weights from B to A. For reference, the faint blue line shows the inverse of the $y = x^2$ relation.

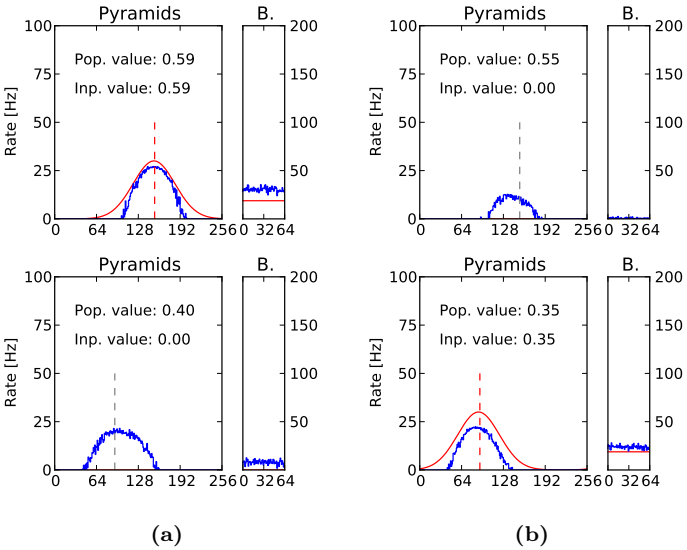


FIGURE 10.3. Inference in a network trained with 1D-inputs. The training inputs followed a quadratic relation ($y = x^2$), which led to the weight matrices shown in Figure 10.2. **(a,b)** Input (red curve) and settled activity (blue curve) in layer A (top) and layer B (bottom). The red dashed lines (vertical) show the encoded input value, and the gray dashed lines show, with respect to the relation, which value should be inferred. **(a)** Inference of B when the only external input (X) is fed to layer A. This is like computing the square of X. **(b)** Inference of A when the only external input (Y) is fed to layer B. This is like computing the square root of Y.

A straight-forward answer is to use them to compute either the square or the square root of some scalar value v. This can be done by encoding v in population X (for computing the square) or population Y (for computing the square root). Figure 10.3 shows how the network performs on such inference tasks.

If not only one, but both input populations are activated by population encoded values v_1 and v_2, interesting network dynamics can be observed. Figure 10.4 shows that a fairly unspecific (wide) activation of layer X can lead to very different, settled activation patterns in A.

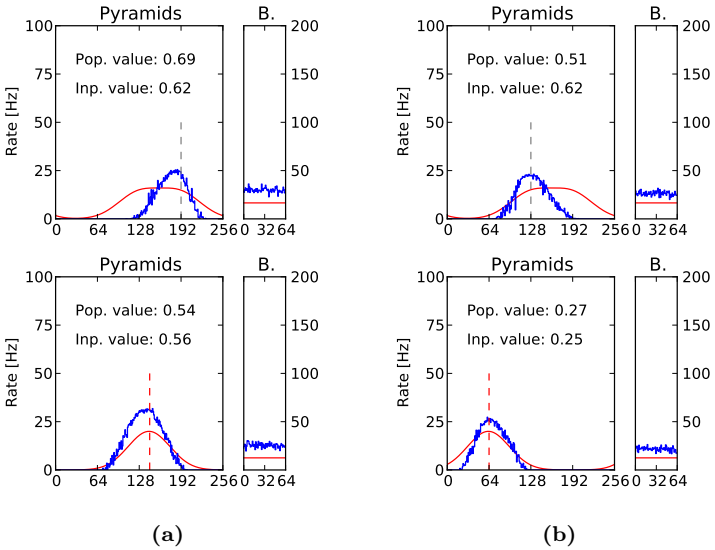


FIGURE 10.4. Biased decisions (or cue-integration) in a network trained with 1D-inputs. The training inputs followed a quadratic relation ($Y = X^2$), which led to the weight matrices shown in Figure 10.2. **(a,b)** Input (red curve) and settled activity (blue curve) in layer A (top) and layer B (bottom). The red dashed lines (vertical) show the encoded input value, while the gray dashed lines show, with respect to the learned relation, which value the given input would correspond to in the opposite layer. **(a)** Layer A "decides", influenced by the activity in layer B, that an activation encoding a value around 0.7 is better in line with the learned relation. **(b)** Now, after the input to layer B changed from 0.56 to 0.25, layer A "decides" differently. With this very different input coming from layer B an activation of A encoding a value around 0.5 is in better agreement with the learned relation.

The difference in how A "decides" to be activated comes through the afferent input of layer B. Layer B gives some additional input to A, which effectively breaks the symmetry of the equilibrated lateral inputs in A. We say that A performs a "biased decision". Note that the settled network response is in both examples in good agreement with the learned relation.

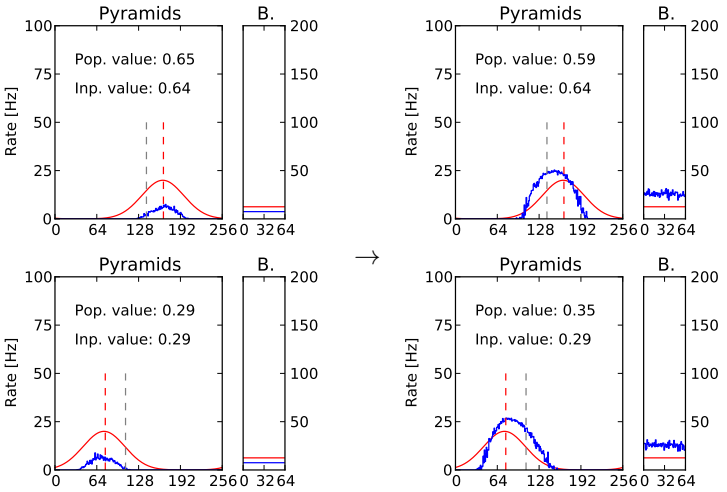


FIGURE 10.5. Cue integration in a network trained with 1D-inputs. The training inputs followed a quadratic relation ($Y = X^2$), which led to the weight matrices shown in Figure 10.2. Input (red curves) and settled activity (blue curves) in layer A (top row) and layer B (bottom row). The red dashed lines (vertical) show the encoded input value, while the gray dashed lines show, with respect to the relation, which value the given input would correspond to in the opposite layer. The left side shows the layer activities right after stimulus onset. The right side shows the settled activities. Input patterns X and Y, which are fed to A and B respectively, encode the values $X = 0.64$ and $Y = 0.29$. Note that these values are not in line with the learned relation. The network, influenced by these conflicting inputs, shifts the lumps of activation in both layers to be more in line with the learned relation.

If, on the other hand, the inputs given in X and Y do not satisfy the learned relation, the settled activities in layer A and B will find a compromise between the currently fed data and a nearby datapoint satisfying the learned relation. We call this "cue integration", because the influences of both input cues are integrated with the learned relation.

Figure 10.5 shows such a case. How far the settled activities move from the given input values toward a nearby learned datapoint satisfying the learned relation can be controlled by the amplitude (the strength) of the fed inputs.

In similar scenarios, where the input amplitudes in X and Y are not equal (as opposed to Figure 10.5 where they are actually the same), the cue integrated compromise would be closer to the stronger of the two inputs. This, as we will see below in Section 10.4.2, is an important property of trained networks.

10.3.1.2. *Sampling a Nonlinear Relations.* As we have mentioned before, the training data was sampled uniformly at random from all pairs (v, v^2) for $v \in [0, 1[$. This implies that the distribution of the total input given to the pyramidal cells in populations A and B during the training phase is non uniform. To see this imagine that values v are unform samples. This implies that v^2 is not uniform in $[0, 1]$. The same argument also applies the other way round and we can conclude that v's and v^2's cannot both be sampled uniformly at random at the same time.

This non-uniformity can cause problems for learning smooth connection matrices. The two most important factors to prevent clustering of weights in the weight matrices are the homeostatic activity regulation (see Sections 9.2.1 and 10.2.1), and a relatively low value for k_{lrn}, so that only input driven, early network responses are learned (see line 29 in Function **learnRelation**() on page 143 and discussion in Section 9.2.2).

Another influential factor is the total strength of all recurrent and afferent connections. The stronger these connections are, the more likely it is that weights begin to cluster during learning.

Although our goal was to learn smooth weight distributions, one might also like to learn clustered weight matrices. While the dynamics of our results lead to continuous activation dynamics, clustered weight matrices would lead to discrete activation dynamics. Such a network would always settle in one of several well defined, discrete states.

10.3.2. Learning a Relation Between Two-Dimensional Inputs. Having seen how the network learned a nonlinear relation between two scalar values, we will now learn a relation defined on two scalar value pairs.

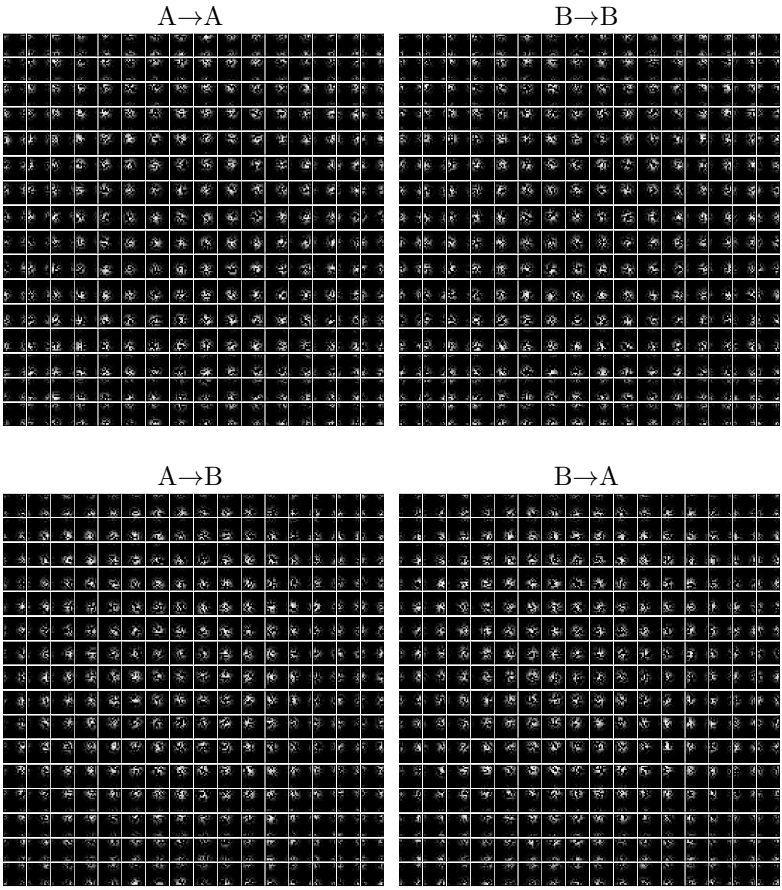


FIGURE 10.6. Learned lateral (topology) connections and afferent (relation) connections. The weights are color coded with white encoding the strongest weight and black encoding zero. Since the topology of the inputs fed to the network in this example was 2-dimensional we have regrouped the weights so that they reflect the topology of the input space. (See Figure 9.5 for further explanation of this visualization, and Section 10.2.3 for details on the relation presented to the network.) **(A→A)** The weights that learned the topology of the input X to layer A. **(B→B)** The weights that learned the topology of the input Y to layer B. **(A→B)** The weights that learned the values of B that correspond to values in A. **(B→A)** The weights that learned the values of A that correspond to values in B.

To demonstrate that coupled RCNs can learn relations of such higher dimensional input encodings, we will learn the relation: $(y_1, y_2) = (1 - x_1, 1 - x_2)$.

Figure 10.6 shows the learned 4D weight tensors using the type of plots we introduced in Chapter 9. By looking at the learned connections we can see that (i) the recurrent connections learned the two-dimensional input topologies well, and (ii) the afferent connections are in-line with the learned relation.

As we did for the previous example we again want to look at the computational abilities of the trained network.

10.3.2.1. *Making Use of the Learned Relation.* Here we basically repeat the same experiments we performed for the first example, the nonlinear, squared relation $y = x^2$, but now in the two-dimensional context of the inversion relation.

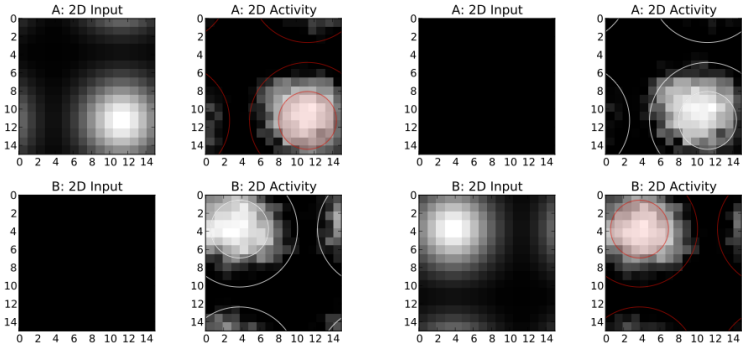


FIGURE 10.7. Inference in a network trained with 2D-inputs. The 2D-encoded training inputs followed the "inversion relation" $(y_1, y_2) = (1-x_1, 1-x_2)$, which led to the weight matrices shown in Figure 10.6. **(a,b)** Input (left column) and settled activity (right column) in layer A (top row) and layer B (bottom row). Red and white circles show the location of the encoded input pattern and the corresponding location in the opposite layer with respect to the inversion relation. **(a)** Inference of B when the only external input $(X = (x_1, x_2))$ is fed to layer A. **(b)** Inference of A when the only external input $(Y = (y_1, y_2))$ is fed to layer B.

Figure 10.7 shows both directions of an inference task. By giving external input to either A or B we can infer an activity pattern in the other layer that satisfies the learned inversion relation defined in Section 10.2.3.

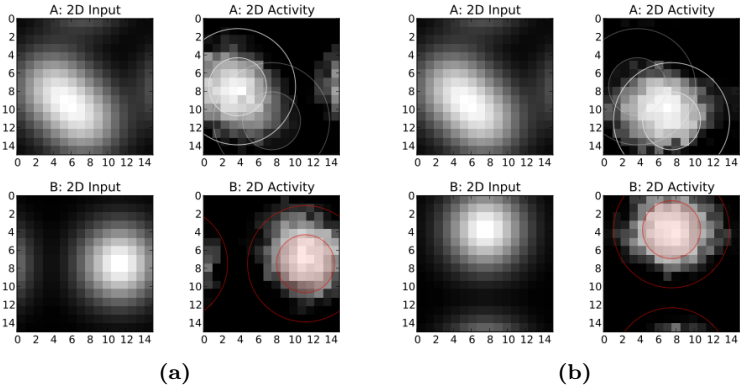


FIGURE 10.8. Biased decisions in a network trained with 2D-inputs. The 2D-encoded training inputs followed the "inversion relation" $(y_1, y_2) = (1 - x_1, 1 - x_2)$, which led to the weight matrices shown in Figure 10.6. **(a,b)** Input (left column) and settled activity (right column) in layer A (top row) and layer B (bottom row). Red and white circles show the location of the encoded input pattern and the corresponding location in the opposite layer with respect to the inversion relation. Input to layer A is potentially activating a large region. Depending on the input given to layer B the actual region of activation can be influenced. (The network makes a "decision" for an activation pattern that is, with respect to the learned relation, more likely.) **(a)** Influenced by the input $\mathbf{v_Y} = (0.75, 0.5)$ to B, the network "decides" that an activation of A encoding a value around $(0.25, 0.5)$ is likely. **(b)** Influenced by the input $\mathbf{v_Y} = (0.5, 0.25)$ to B, the network chooses an activation of A encoding a value around $(0.5, 0.75)$.

Figure 10.8 shows, similar to Figure 10.4, that diffuse activation in one layer can be tightened considerably by feeding some tie-breaking input to the second layer. Different amplitudes of the activities fed to either of the two layers can slightly change the settled network state in favor

of the stronger input. (The notion of stronger is not very well defined and is in general a combination of input width and amplitude.)

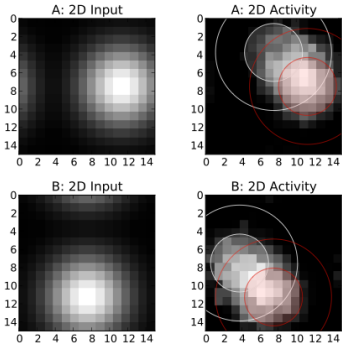


FIGURE 10.9. Cue integration in a network trained with 2D-inputs. The 2D-encoded training inputs followed the relation $(y_1, y_2) = (1 - x_1, 1 - x_2)$, which led to the weight matrices shown in Figure 10.6. Input (left column) and settled activity (right column) in layer A (top row) and layer B (bottom row). Red and white circles show the location of the encoded input pattern and the corresponding location in the opposite layer with respect to the learned input relation. Inputs values to layer A and B ($\mathbf{v_X} = (0.75, 0.5)$ and $\mathbf{v_Y} = (0.5, 0.75)$) are not satisfying the learned relation. The network, influenced by these conflicting opinions, shifts the lumps of activation in both layers toward a nearby data point that follows the learned relation (roughly at $\mathbf{v_A} = (0.62, 0.38)$ and $\mathbf{v_B} = (0.38, 0.62)$).

If both inputs are fairly sharp (focused), but do not satisfy the trained relation, the networks integrates the conflicting cues with the learned knowledge, as can be seen in Figure 10.9.

A slightly different network computation is performed when inputs to A and B are both very wide. If one of the two RCNs would be activated by a tight lump of activation we would see the biased decision dynamics we have seen in Figure 10.8, but since both layers can now pick from a wide range of potential states, the network will pick the one that best satisfies both the inputs and the learned relation, as shown in Figure 10.10. This works because the afferent connections will reinforce mutually consistent

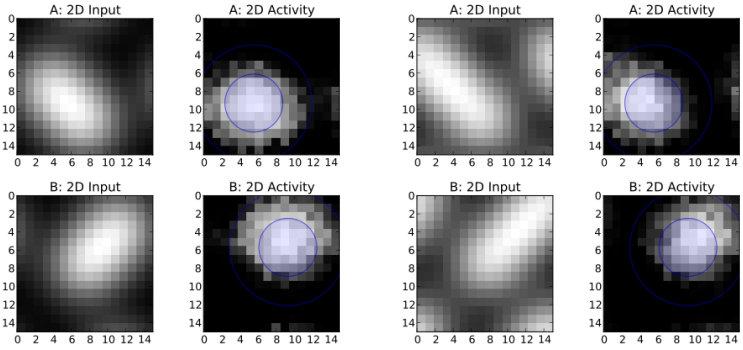


FIGURE 10.10. Constraint satisfaction in a network trained with 2D-inputs. The 2D-encoded training inputs followed the "inversion relation" $(y_1, y_2) = (1 - x_1, 1 - x_2)$, which led to the weight matrices shown in Figure 10.6. **(a,b)** Input (left column) and settled activity (right column) in layer A (top row) and layer B (bottom row). Blue circles show the only solution compatible with both, the given input and the learned relation. **(a)** Inputs to layer A and B are more diffuse than the input lumps used for learning. Blue circles mark the spots that match these diffused inputs and are in-line with the learned relation. **(b)** Inputs to layer A and B are spread over large parts of the individual layers, effectively providing constraints on X and Y. Blue circles mark the spot that matches both, the given input regions and the learned relation.

parts more strong than all other places in A and B. If the broad inputs are thought of as constraints, the network is performing a constraint satisfaction task.

10.3.3. Relation Learning in Noisy Environments. As mentioned in the introduction we want to show that the proposed network is robust to noise in the training data.

We repeat both learning examples presented above. This time we add a significant amount of noise to all input patterns in both, the training and test data.

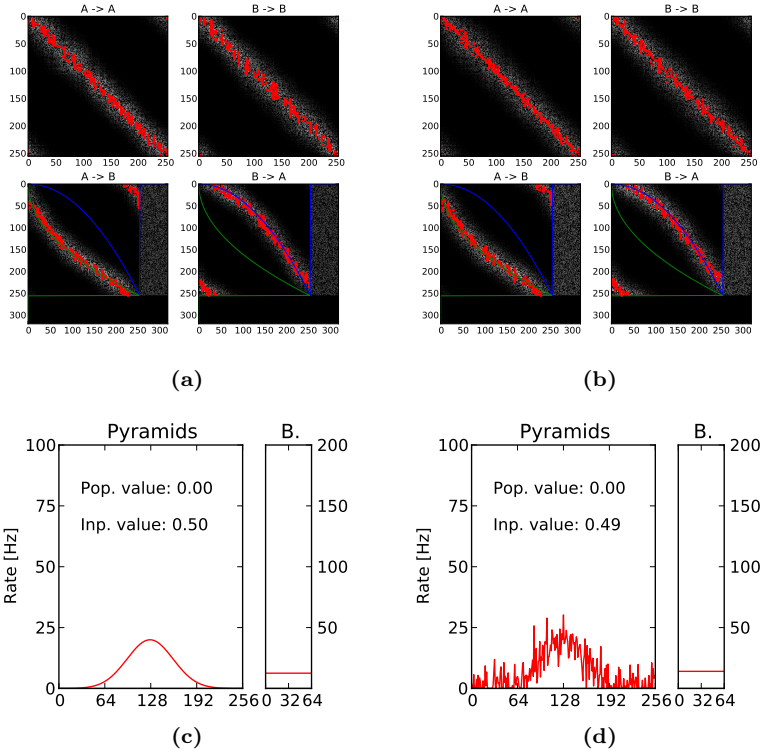


FIGURE 10.11. Smooth vs. noisy training data. **(a,b)** Learned lateral (topology) connections and afferent (relation) connections. The weights are color coded with white encoding the strongest weight and black encoding zero. Red dots indicate the largest weight in the corresponding matrix row. (If there are multiple maxima, the red dot marks the first one occurring in each row.) In both cases we used a quadratic relation to learn the weight matrices shown above. See text for further details. **(a)** Weights learned with smooth, continuously encoded inputs as shown in **(c)**. **(b)** Weights learned with noisy inputs as shown in **(d)**.

Similar to the results in Chapter 9, where we have shown that single RCNs can cope with fairly high levels of noise, the performance for coupled RCNs is also convincing.

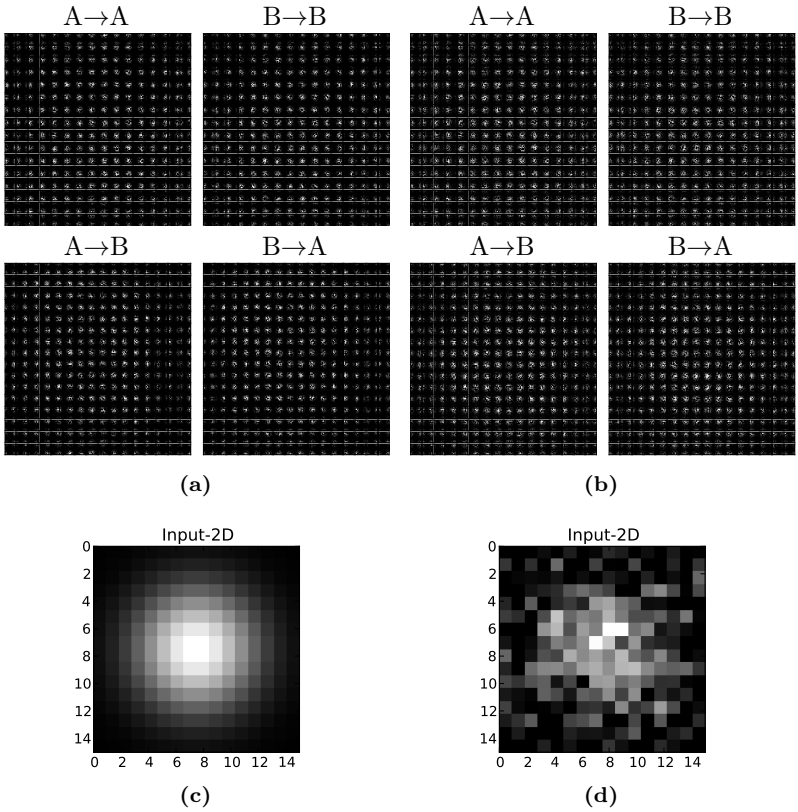


FIGURE 10.12. Smooth vs. noisy 2D training data. **(a,b)** Learned lateral (topology) connections and afferent (relation) connections. The weights are color coded with white encoding the strongest weight and black encoding zero. (See also Figure 9.5 for further explanation of this visualization.) **(a)** Weights learned with smooth, continuously encoded 2D-inputs as shown in **(c)**. **(b)** Weights learned with noisy 2D-inputs as shown in **(d)**.

10.3.3.1. *The Noise Model.* Although the network can cope with different types of noise, the only results we show here are for a relatively simple, gaussian noise model. After an input pattern is created, as explained in Section 9.2.3, we use the peak input rate $r_{\max}$ and perturb each rate r_i to become

$$r_{i,\text{noisy}} = \max(0.0, \mathcal{N}(r_i, \frac{1}{4} r_{\max})) \tag{10.1}$$

The homeostasis (HAR) used in our model enables the network to deal with a certain amount of non-symmetric and or dependent noise. If, for example, each cell would be exposed to a noise source that does not depend on the current input but rather on the identity of this very cell, the input topology can still be learned. (Data not shown.)

10.3.3.2. *Results with Noisy Environments.* Figure 10.11 and Figure 10.12 compare the weight matrices we have learned before with non-noisy inputs with the weight matrices learned in noisy environments.

The results are so similar that the matrices are barely distinguishable by eye. Not only do the learned connections look almost the same, but if we test the network by feeding the noisy equivalents to the test instances used in Sections 10.3.1.1 and 10.3.2.1, the observable dynamics are also very similar.

Figures 10.13 and 10.14 show these results in the following order: (*i*) inference, (*ii*) biased decisions, (*iii*) cue integration for both example relations, and (*iv*) constraint satisfaction, for the noisy relation on 2D-inputs.

10.4. Discussion

In Chapter 9 we saw how an untrained, tabula rasa like RCN can learn the topology of population encoded inputs. By learning the input topology, the network obtained a notion of what typical input patterns tend to look like. This enabled the trained RCN to let multiple input values compete with each other.

In this chapter we have extended the network by a second RCN layer. Both layers are bidirectionally coupled. The coupling is modeled by independent, excitatory random connections, as shown in Figure 10.1. These connections are plastic, being learned in precisely the same way we learn the recurrent connections within the RCN layers.

It is desirable to use the same learning procedure for the newly added connections, because in this way a cell does not have to distinguish the incoming connections by the connections' origin. The neuron uses a single, local rule.

By feeding different input streams we showed that the network is able to learn the hidden relation and the input encoding (the input topology) at the same time. We did not have to artificially introduce an ordering such as first learning the input topology and then learning the relation.

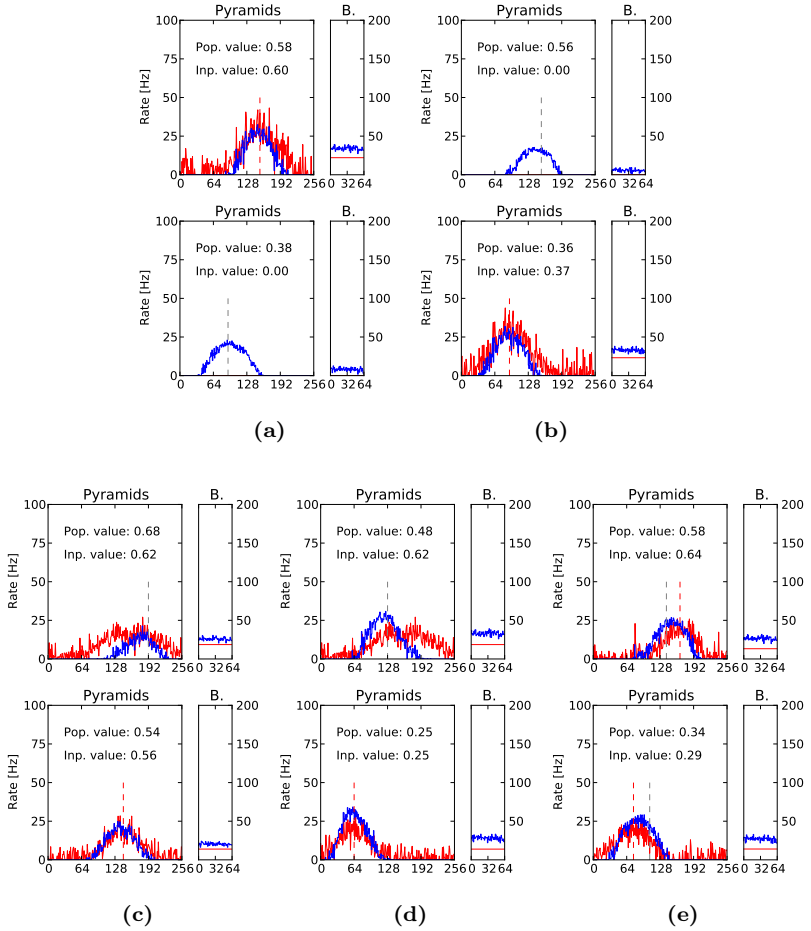


FIGURE 10.13. Settled responses of the network when trained with noisy 1D-inputs drawn from a quadratic relation (see text for details). Learned weights are shown in Figure 10.11. **(a-e)** Noisy inputs and corresponding outputs of RCNs A (top) and B (bottom). The red dashed lines (vertical) show the encoded input value, while the gray dashed lines show, with respect to the relation, which value the given input would correspond to in the opposite layer. **(a)** Noisy equivalent to Figure 10.3(a). **(b)** Noisy equivalent to Figure 10.3(b). **(c)** Noisy equivalent to Figure 10.4(a). **(d)** Noisy equivalent to Figure 10.4(b). **(e)** Noisy equivalent to Figure 10.5.

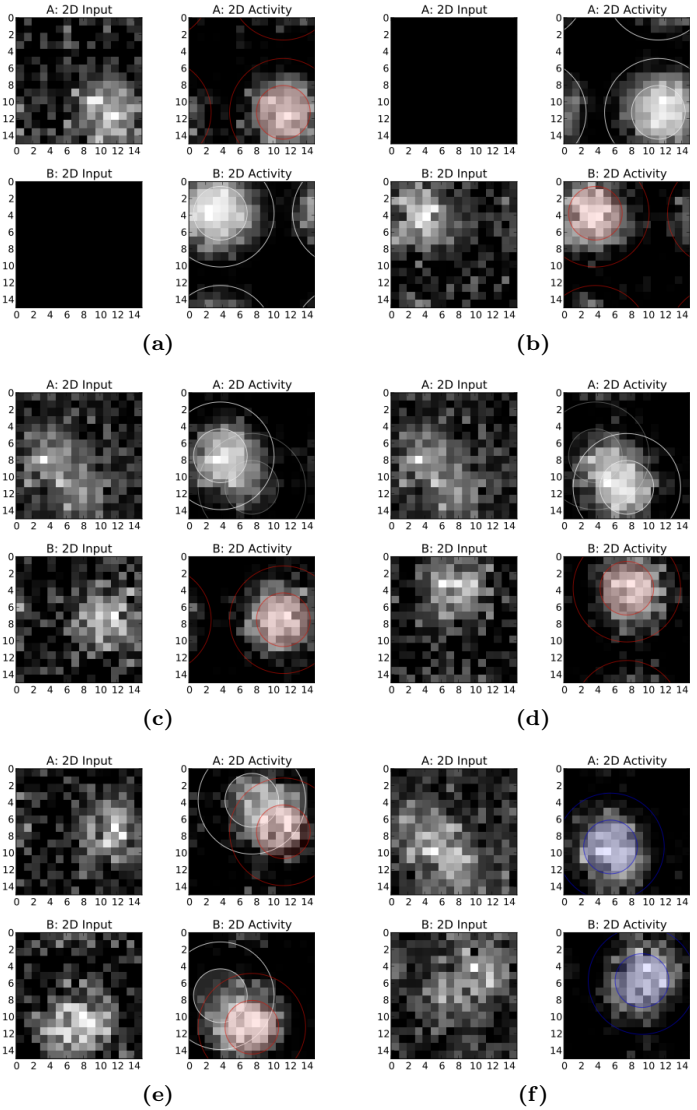


FIGURE 10.14. Settled responses of the network when trained with noisy 2D-inputs drawn from the "inversion relation". Learned weights are shown in Figure 10.12. **(a-f)** Noisy inputs and corresponding outputs of RCNs A (top row) and B (bottom row). Colored circles are as in previous figures. **(a-f)** Noisy equivalents to Figures 10.7(a), 10.7(b), 10.8(a), 10.8(b), 10.9, and 10.10.

The ability to learn everything at the same time is a desirable feature for many learning tasks, because we do not need any higher order process that takes care of switching between multiple learning rules or phases – in our example the network just learns what it has to learn using a set of simple, local, biologically plausible principles.

Last but not least we have shown that the proposed network can deal with a significant amount of noise. Because the training data consists of several hundred example inputs, the ability to deal with symmetric noise follows directly from the dynamics. Although we did not show these results here we want to mention that training sessions that included biased, non-symmetric noise also led to good results.

We did not include any quantitative analysis of how different noise models reduce the learnability or usability of this approach. Such evaluations are left for the future.

The flexibility to adapt to different input topologies (input encodings), the robustness to noise, and the fact that coupled RCNs only consist of locally operating, biological concepts makes this kind of network an interesting model for cortical computation.

10.4.1. Learning Relations That are not Functions. Both example relations used in this chapter are functions. How would the proposed network deal with relations that is not a function? In Chapter 3 we trained non-invertible functions and showed that learning in such networks can work well. All examples contained in Chapter 3 work well in the network given in this chapter (data not shown).

Next we want to discuss some slightly more extreme cases than the ones mentioned before.

Let us consider a relation like $x \leq y$. Giving one value for either x or y does not restrict the possibilities for the second variable to only one (or maybe two or three values as in Chapter 3), but offers a continuous range of permitted values.

Since the data we train our network with would always show random but valid examples, the recurrent connections can learn the used input topology without problems. However, the afferent connection weights in this case need to spread or distribute across a much larger area, basically covering the entire range of permitted values.

Although the network we described does a fairly good job learning precisely this weight distribution, the results are less useful than for the

other examples before. In large parts this problem arises by reason of the overly simplistic learning rule that keeps the sum of all weights in each weight matrix constant. We believe this problem can be addressed by finding a better and perhaps more realistic normalization mechanism.

Learning and using such diffuse weight matrices for inference, decision making, cue-integration, and solving constraint-satisfaction problems will be the subject of future research. Preliminary results show interesting and useful dynamics, and we believe that they will prove to be useful in building large scale computational models and cognitive reasoning systems.

10.4.2. A Probabilistic Interpretation of Coupled RCNs. In Section 9.4.1 on page 133 we have already given a probabilistic interpretation of single RCNs. There we argued that a population encoded input pattern α as well as the output activity σ can be interpreted as being a sort of restricted probability distribution.

Another interesting observation of Section 9.4.1 is that learning the structure of typical inputs ($\mathcal{C}$) biased the output activity patterns σ. We argued that the dynamics of a trained RCN can be understood as finding an energy minimum of an energy or error function like

$$\mathcal{E}(\sigma, \alpha) = \sum_i |\alpha_i - \sigma_i|^{k_1} + \kappa \cdot \sum_{i,j} |\mathcal{C}_{i,j} - f(\sigma_i, \sigma_j)|^{k_2}. \tag{10.2}$$

We concluded that the activities in trained RCNs settle qualitatively to an output activity σ that is approximating

$$\sigma^* = \underset{\sigma}{\operatorname{argmin}} \left(\mathcal{E}(\sigma, \alpha)\right). \tag{10.3}$$

In this chapter we used networks of two coupled RCNs. We will now extend the probabilistic interpretation of Chapter 9 to include also the second RCN layer and the mutual interaction between the two layers in the network. Consider

$$\mathcal{E}_{\mathrm{RL}}(\sigma_1, \alpha_1, \sigma_2, \alpha_2) = \mathcal{E}(\sigma_1, \alpha_1) + \mathcal{E}(\sigma_2, \alpha_2) + \kappa_2 \cdot \sum_{i,j} |\mathcal{R}_{i,j} - g(\sigma_i, \sigma_j)|^{k_3}, \tag{10.4}$$

with k_3 being a constant controlling the dynamics of the affected summands in $\mathcal{E}_{\mathrm{RL}}$, and κ_2 being a constant tradeoff factor. The first two terms are given in Eq. 10.2, while the last term punishes outputs that are not compatible with the learned relation.

One possible formalization for g, the similarity measure for output activities, is

$$g(x, y) = 1 - |x - y|. \tag{10.5}$$

As for Equation 9.13 we assumed that desired outputs (σ_i) and the activity of input neurons (α_j) are given as values $\in [0, 1]$.

Hence, $\mathcal{E}_{\mathrm{RL}}$ formalizes a tradeoff between (*i*) the input patterns α_1 and α_2, (*ii*) the system's learned topological knowledge $\mathcal{C}$ within each layer, and (*iii*) the system's learned relational knowledge $\mathcal{R}$ between those.

The dynamics of two trained, coupled RCNs can now, similarly to how we have seen previously in Eq. 9.14 on page 135, be understood as finding an energy minimum by relaxing to output activities (σ_1, σ_2) that are roughly

$$(\sigma_1^*, \sigma_2^*) = \underset{\sigma_1, \sigma_2}{\operatorname{argmin}} \left(\mathcal{E}_{\mathrm{RL}}(\sigma_1, \alpha_1, \sigma_2, \alpha_2) \right). \tag{10.6}$$

In this context we see all the abilities of the learned network, *inference*, *decision making*, *cue-integration*, and *signal restoration* simply as the network's attempt to relax to a consistent, minimal energy state.

List of Tables

Bibliography

[AB97] D J Amit and N Brunel, *Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex.*, Cerebral cortex (New York, NY : 1991) **7** (1997), no. 3, 237–252.

[Abe82] M Abeles, *Local cortical circuits: An electrophysiological study*, Springer-Verlag (Berlin and New York), 1982.

[Abr02] Tara H Abraham, *(Physio)logical circuits: the intellectual origins of the McCulloch-Pitts neural networks.*, vol. 38, Dibner Institute for the History of Science and Technology, MIT, USA., 2002.

[AM00] S Aji and R McEliece, *The generalized distributive law*, Information Theory (2000).

[Ama77] S Amari, *Dynamics of pattern formation in lateral-inhibition type neural fields*, Biological Cybernetics (1977).

[AN00] L F Abbott and S B Nelson, *Synaptic plasticity: taming the beast.*, Nature Neuroscience **3 Suppl** (2000), 1178–1183.

[ASLB07] Collins Assisi, Mark Stopfer, Gilles Laurent, and Maxim Bazhenov, *Adaptive regulation of sparseness by feedforward inhibition.*, Nature Neuroscience **10** (2007), no. 9, 1176–1184.

[BB03] JM Bower and D Beeman, *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*, Free Internet Edition, 2003.

[BBG04] J Brown, D Bullock, and S Grossberg, *How laminar frontal cortex and basal ganglia circuits interact to control planned and reactive saccades*, Neural Networks (2004).

[BBK06] A Barco, CH Bailey, and ER Kandel, *Common molecular mechanisms in explicit and implicit memory*, Journal of neurochemistry **97** (2006), no. 6, 1520–1533.

[BC83] CH Bailey and M Chen, *Morphological basis of long-term habituation and sensitization in Aplysia*, Science (New York, NY) **220** (1983), no. 4592, 91–93.

[BC88] ———, *Morphological basis of short-term habituation in Aplysia*, The Journal of neuroscience : the official journal of the Society for Neuroscience **8** (1988), no. 7, 2452–2459.

[BCHS95] E H Buhl, S R Cobb, K Halasy, and P Somogyi, *Properties of unitary IPSPs evoked by anatomically identified basket cells in the rat hippocampus*, The European journal of neuroscience **7** (1995), no. 9, 1989–2004.

[BD11] Martin Boerlin and Sophie Deneve, *Spike-Based Population Coding and Working Memory*, PLoS Computational Biology **7** (2011), no. 2, e1001080.

[BDM04] T Binzegger, R Douglas, and K Martin, *A Quantitative Map of the Circuit of Cat Primary Visual Cortex*, Journal of Neuroscience (2004).

[BDMK91] O Bernander, R J Douglas, K A Martin, and C Koch, *Synaptic background activity influences spatiotemporal integration in single pyramidal cells*, Proceedings of the National Academy of Sciences of the United States of America **88** (1991), no. 24, 11569–11573.

[BG92] E Bartfeld and A Grinvald, *Relationships Between Orientation-Preference Pinwheels, Cytochrome Oxidase Blobs, and Ocular-Dominance Columns in Primate Striate Cortex*, Proceedings of the National Academy of Sciences (1992).

[BGC03] Michael Beierlein, Jay R Gibson, and Barry W Connors, *Two dynamically distinct inhibitory networks in layer 4 of the neocortex*, Journal of Neurophysiology **90** (2003), no. 5, 2987–3000.

[BK93] CH Bailey and ER Kandel, *Structural changes accompanying memory storage*, Annual Review of Physiology **55** (1993), 397–426.

[BMS07] Y Banitt, K A C Martin, and I Segev, *A Biologically Realistic Model of Contrast Invariant Orientation Tuning by Thalamocortical Synaptic Depression*, Journal of Neuroscience **27** (2007), no. 38, 10230–10239.

[Bol01] Béla Bollobás, *Random graphs*, second edition ed., Cambridge Univ. Press, 2001.

[BPV+11] Daniel Brüderle, Mihai A Petrovici, Bernhard Vogginger, Matthias Ehrlich, Thomas Pfeil, Sebastian Millner, Andreas Grübl, Karsten Wendt, Eric Müller, Marc-Olivier Schwartz, Dan Husmann de Oliveira, Sebastian Jeltsch, Johannes Fieres, Moritz Schilling, Paul Müller, Oliver Breitwieser, Venelin Petkov, Lyle Muller, Andrew P Davison, Pradeep Krishnamurthy, Jens Kremkow, Mikael Lundqvist, Eilif Muller, Johannes Partzsch, Stefan Scholze, Lukas Zühl, Christian Mayr, Alain Destexhe, Markus Diesmann, Tobias C Potjans, Anders Lansner, René Schüffny, Johannes Schemmel, and Karlheinz Meier, *A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems.*, Biological Cybernetics **104** (2011), no. 4-5, 263–296.

[BRC+07] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P Davison, Sami el Boustani, and Alain Destexhe, *Simulation of networks of spiking neurons: a review of tools and strategies.*, Journal of Computational Neuroscience **23** (2007), no. 3, 349–398.

[Bru00] N Brunel, *Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons.*, Journal of Computational Neuroscience **8** (2000), no. 3, 183–208.

[BS91] V Braitenberg and A Schütz, *Peters' Rule and White's Exceptions*, Anatomy of the cortex (1991), 109–111.

[BS06] Randy Bruno and Bert Sakmann, *Cortex Is Driven by Weak but Synchronously Active Thalamocortical Synapses*, Science (New York, NY) **312** (2006), no. 5780, 1622–1627.

[BSF07] Joseph M Brader, Walter Senn, and Stefano Fusi, *Learning real-world stimuli in a neural network with spike-driven synaptic dynamics.*, Neural Computation **19** (2007), no. 11, 2881–2912.

[BSHS96] E H Buhl, T Szilágyi, K Halasy, and P Somogyi, *Physiological properties of anatomically identified basket and bistratified cells in the CA1 area of the rat hippocampus in vitro.*, Hippocampus **6** (1996), no. 3, 294–305.

[BSW93] A Baranyi, M B Szente, and C D Woody, *Electrophysiological characterization of different types of neurons recorded in vivo in the motor cortex of the cat. II. Membrane parameters, action potentials, current-induced voltage responses and electrotonic structures*, Journal of Neurophysiology **69** (1993), no. 6, 1865–1879.

[Bur06] A N Burkitt, *A Review of the Integrate-and-fire Neuron Model: I. Homogeneous Synaptic Input*, Biological Cybernetics **95** (2006), no. 1, 1–19.

[Buz06] Gyorgy Buzsaki, *Rhythms of the Brain*, 1 ed., Oxford University Press, USA, August 2006.

[BW03] N Brunel and X Wang, *What determines the frequency of fast network oscillations with irregular neural discharges? I. ...*, Journal of Neurophysiology (2003).

[BZN01] W Bair, E Zohary, and W T Newsome, *Correlated firing in macaque visual area MT: time scales and relationship to behavior.*, The Journal of neuroscience : the official journal of the Society for Neuroscience **21** (2001), no. 5, 1676–1697.

[BZSF97] W Bosking, Y Zhang, B Schofield, and D Fitzpatrick, *Orientation Selectivity and the Arrangement of Horizontal Connections in Tree Shrew Striate Cortex*, Journal of Neuroscience (1997).

[CB04] M Cook and J Bruck, *Networks of Relations for Representation, Learning, and Generalization*, Proceedings of the Fourth International Conference on ... (2004).

[CGP82] B W Connors, M J Gutnick, and D A Prince, *Electrophysiological properties of neocortical neurons in vitro*, Journal of Neurophysiology **48** (1982), no. 6, 1302–1320.

[CH11] Matteo Carandini and David J Heeger, *Normalization as a canonical neural computation.*, Nature Reviews Neuroscience (2011).

[CHK83] TJ Carew, R D Hawkins, and ER Kandel, *Differential classical conditioning of a defensive withdrawal reflex in Aplysia californica*, Science (New York, NY) **219** (1983), no. 4583, 397–400.

[CJK10] M Cook, F Jug, and C Krautz, *Unsupervised learning of relations*, Artificial Neural Networks–ICANN 2010 (2010).

[CJK11] ———, *Neuronal Projections Can Be Sharpened by a Biologically Plausible Learning Mechanism*, Artificial Neural Networks–ICANN 2011 (2011).

[CLR+11] Xiaowei Chen, Ulrich Leischner, Nathalie L Rochefort, Israel Nelken, and Arthur Konnerth, *Functional mapping of single spines in cortical neurons in vivo.*, Nature **475** (2011), no. 7357, 501–505.

[Cri89] F Crick, *The recent excitement about neural networks*, Nature **337** (1989), 129–132.

[DA01] Peter Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, 1st ed., The MIT Press, December 2001.

[DC06] Alain Destexhe and Diego Contreras, *Neuronal computations with stochastic network states.*, Science (New York, NY) **314** (2006), no. 5796, 85–90.

[DCM09] N M Da Costa and K A C Martin, *Selective Targeting of the Dendrites of Corticothalamic Cells by Thalamic Afferents in Area 17 of the Cat*, Journal of Neuroscience **29** (2009), no. 44, 13919–13928.

[DCM11] ———, *How thalamus connects to spiny stellate cells in the cat's visual cortex*, Preprint (2011), 1–45.

[DG02] M Diesmann and MO Gewaltig, *NEST: An environment for neural systems simulations*, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (2002), 43.

[DGA99] M Diesmann, M O Gewaltig, and A Aertsen, *Stable propagation of synchronous spiking in cortical neural networks.*, Nature **402** (1999), no. 6761, 529–533.

[DGFM03] Paolo Del Giudice, Stefano Fusi, and Maurizio Mattia, *Modelling the formation of working memory with networks of integrate-and-fire neurons connected by plastic synapses.*, Journal of physiology, Paris **97** (2003), no. 4-6, 659–681.

[DLP99] Sophie Deneve, Peter E Latham, and Alexandre Pouget, *Reading population codes: a neural implementation of ideal observers*, Nature Neuroscience **2** (1999), no. 8, 740.

[DLP01] S Deneve, P Latham, and A Pouget, *Efficient computation and cue integration with noisy population codes*, Nature Neuroscience **4** (2001), no. 8, 826–831.

[DM07] R Douglas and K Martin, *Recurrent neuronal circuits in the neocortex*, Current Biology **17** (2007), no. 13, 496–500.

[Dom06] Peter Ford Dominey, *Encodign Temporal Structure of Language in Recurrent Network of Leaky Integrators.*

[DP89] R A Deisz and D A Prince, *Frequency-dependent depression of inhibition in guinea-pig neocortex in vitro by GABAB receptor feed-back on GABA release*, The Journal of Physiology **412** (1989), 513–541.

[DP92] Y Dan and M M Poo, *Hebbian depression of isolated neuromuscular synapses in vitro.*, Science (New York, NY) **256** (1992), no. 5063, 1570–1573.

[DPH+09] K Dockendorf, I Park, P He, J Principe, and T Demarse, *Liquid state machines and cultured cortical networks: The separation property*, BioSystems **95** (2009), no. 2, 90–97.

[DRFS01] A Destexhe, M Rudolph, J M Fellous, and T J Sejnowski, *Fluctuating synaptic conductances recreate in vivo-like activity in neocortical neurons.*, Neuroscience **107** (2001), no. 1, 13–24.

[DTGG02] Eric Dégenètais, Anne-Marie Thierry, Jacques Glowinski, and Yves Gioanni, *Electrophysiological properties of pyramidal neurons in the rat prefrontal cortex: an in vivo intracellular recording study*, Cerebral cortex (New York, NY : 1991) **12** (2002), no. 1, 1–16.

[EBK+10] Alexander S Ecker, Philipp Berens, Georgios A Keliris, Matthias Bethge, Nikos K Logothetis, and Andreas S Tolias, *Decorrelated neuronal firing in cortical microcircuits.*, Science (New York, NY) **327** (2010), no. 5965, 584–587.

[EHM08] J Eppler, M Helias, and E Muller, *PyNEST: a convenient interface to the NEST simulator*, Frontiers in ... (2008).

[ER60] P Erdős and A Rényi, *On the Evolution of Random Graphs*, Publ. Math. Inst. Hung. Acad. Sci. (1960).

[EWL+91] O Ekeberg, P Wallén, A Lansner, H Tråvén, L Brodin, and S Grillner, *A computer based model for realistic simulations of neural networks. I. The single neuron and synaptic interaction.*, Biological Cybernetics **65** (1991), no. 2, 81–90.

[EZ78] D Essen and S Zeki, *The topographic organization of rhesus monkey prestriate cortex*, The Journal of Physiology (1978).

[FFdSCB10] Marco Fuenzalida, David Fernández de Sevilla, Alejandro Couve, and Washington Buño, *Role of AMPA and NMDA receptors and back-propagating action potentials in spike timing-dependent plasticity.*, Journal of Neurophysiology **103** (2010), no. 1, 47–54.

[FJ01] G Forney Jr, *Codes on graphs: normal realizations*, Information Theory (2001).

[FM00] D Fricker and R Miles, *EPSP amplification and the precision of spike timing in hippocampal neurons*, Neuron **28** (2000), no. 2, 559–569.

[FvE91] D Felleman and D van Essen, *Distributed Hierarchical Processing in the Primate Cerebral Cortex*, Cerebral Cortex **1** (1991), no. 1, 1–47.

[FVM99] D Fricker, J A Verheugen, and R Miles, *Cell-attached measurements of the firing threshold of rat hippocampal neurones*, The Journal of Physiology **517 (Pt 3)** (1999), 791–804.

[GA00] Z Gil and Y Amitai, *Evidence for proportional synaptic scaling in neocortex of intact animals*, Neuroreport **11** (2000), no. 18, 4027–4031.

[GBKP+05] Guillermo González-Burgos, Leonid S Krimer, Nadya V Povysheva, Germán Barrionuevo, and David A Lewis, *Functional properties of fast spiking interneurons and their synaptic connections with pyramidal cells in primate dorsolateral prefrontal cortex.*, Journal of Neurophysiology **93** (2005), no. 2, 942–953.

[GBKU+04] Guillermo González-Burgos, Leonid S Krimer, Nathaniel N Urban, Germán Barrionuevo, and David A Lewis, *Synaptic efficacy during repetitive activation of excitatory inputs in primate dorsolateral prefrontal cortex*, Cerebral cortex (New York, NY : 1991) **14** (2004), no. 5, 530–542.

[GH98] M Galarreta and S Hestrin, *Frequency-dependent synaptic depression and the balance of excitation and inhibition in the neocortex*, Nature Neuroscience **1** (1998), no. 7, 587–594.

[GH01] ———, *Spike transmission and synchrony detection in networks of GABAergic interneurons*, Science (New York, NY) **292** (2001), no. 5525, 2295–2299.

[GK02] Wulfram Gerstner and Werner M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, 2002.

[GKCM82] A P Georgopoulos, J F Kalaska, R Caminiti, and J T Massey, *On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex*, The Journal of neuroscience : the official journal of the Society for Neuroscience **2** (1982), no. 11, 1527–1537.

[GLC+01] B S Gutkin, C R Laing, C L Colby, C C Chow, and G B Ermentrout, *Turning on and off with excitation: the role of spike-timing asynchrony and synchrony in sustained neural activity*, Journal of Computational Neuroscience **11** (2001), no. 2, 121–134.

[Gro99] S Grossberg, *How does the cerebral cortex work? Learning, attention, and grouping by the laminar circuits of visual cortex*, Spatial Vision (1999).

[GWGR03] Wen-Jun Gao, Yun Wang, and Patricia S Goldman-Rakic, *Dopamine modulation of perisomatic and peridendritic inhibition in prefrontal cortex*, The Journal of neuroscience : the official journal of the Society for Neuroscience **23** (2003), no. 5, 1622–1630.

[HC97] M L Hines and N T Carnevale, *The NEURON Simulation Environment*, Neural Computation **9** (1997), no. 6, 1179–1209.

[Heb49] DO Hebb, *The Organization of Behavior: A Neuropsychological Theory*, Wiley (1949).

[Hee92] D J Heeger, *Normalization of cell responses in cat striate cortex*, Visual Neuroscience **9** (1992), no. 2, 181–197.

[HH07] J Heinzle and K Hepp, *A microcircuit model of the frontal eye fields*, Journal of Neuroscience (2007).

[HHSZ03] Carl Holmgren, Tibor Harkany, Björn Svennenfors, and Yuri Zilberter, *Pyramidal cell communication within local networks in layer 2/3 of rat neocortex*, The Journal of Physiology **551** (2003), no. Pt 1, 139–153.

[HM07] S Haeusler and W Maass, *A Statistical Analysis of Information-Processing Properties of Lamina-Specific Cortical Microcircuit Models*, Cerebral Cortex (2007).

[HNPS90a] S Hestrin, R A Nicoll, D J Perkel, and P Sah, *Analysis of excitatory synaptic action in pyramidal cells using whole-cell recording from rat hippocampal slices*, The Journal of Physiology **422** (1990), 203–225.

[HNPS90b] ———, *Analysis of excitatory synaptic action in pyramidal cells using whole-cell recording from rat hippocampal slices*, The Journal of Physiology **422** (1990), 203–225.

[HRB11] Kosuke Hamaguchi, Alexa Riehle, and Nicolas Brunel, *Estimating network parameters from combined dynamics of firing rate and irregularity of single neurons.*, Journal of Neurophysiology **105** (2011), no. 1, 487–500.

[HSN90] S Hestrin, P Sah, and R A Nicoll, *Mechanisms generating the time course of dual component excitatory synaptic currents recorded in hippocampal slices*, Neuron **5** (1990), no. 3, 247–253.

[HT05] Sean Hill and Giulio Tononi, *Modeling sleep and wakefulness in the thalamocortical system.*, Journal of Neurophysiology **93** (2005), no. 3, 1671–1698.

[HW62] D Hubel and T Wiesel, *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*, The Journal of Physiology (1962).

[ID03] Eugene M Izhikevich and Niraj S Desai, *Relating STDP to BCM.*, Neural Computation **15** (2003), no. 7, 1511–1523.

[IH11] Giacomo Indiveri and Timothy K Horiuchi, *Frontiers in Neuromorphic Engineering*, Frontiers in Neuroscience **5** (2011).

[Izh04] Eugene M Izhikevich, *Which model to use for cortical spiking neurons?*, IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council **15** (2004), no. 5, 1063–1070.

[Izh06] ———, *Polychronization: computation with spikes.*, Neural Computation **18** (2006), no. 2, 245–282.

[Kan91] ER Kandel, *Cellular mechanisms of learning and the biological basis of individuality*, Principles of neural science (1991).

[KFL01] F Kschischang, B Frey, and H Loeliger, *Factor graphs and the sum-product algorithm*, IEEE Transactions on Information Theory (2001).

[KHOC07] Theofanis Karayannis, Icnelia Huerta-Ocampo, and Marco Capogna, *GABAergic and pyramidal neurons of deep cortical layers directly receive and differently integrate callosal input*, Cerebral cortex (New York, NY : 1991) **17** (2007), no. 5, 1213–1226.

[KHP+11] Ho Ko, Sonja B Hofer, Bruno Pichler, Katherine A Buchanan, P Jesper Sjöström, and Thomas D Mrsic-Flogel, *Functional specificity of local synaptic connections in neocortical networks.*, Nature **473** (2011), no. 7345, 87–91.

[KO08] Pratap Kumar and Ora Ohana, *Inter- and intralaminar subcircuits of excitatory and inhibitory neurons in layer 6a of the rat barrel cortex*, Journal of Neurophysiology **100** (2008), no. 4, 1909–1922.

[Koh97] Teuvo Kohonen, *Self-organizing maps*, Secaucus, NJ, USA, 1997.

[KRA08] A Kumar, S Rotter, and A Aertsen, *Conditions for Propagating Synchronous Spiking and Asynchronous Firing Rates in a Cortical Network ...*, Journal of Neuroscience (2008).

[KRA10] Arvind Kumar, Stefan Rotter, and Ad Aertsen, *Spiking activity propagation in neuronal networks: reconciling different perspectives on neural coding*, Nature Reviews Neuroscience **11** (2010), no. 9, 615–627.

[Kra12] Christoph Krautz, *From neurons to systems*, Ph.D. thesis, ETH Zurich, 2012.

[KS61] ER Kandel and W A Spencer, *Electrophysiology of hippocampal neurons. II. After-potentials and repetitive firing*, Journal of Neurophysiology **24** (1961), 243–259.

[KSBH94] H Kennedy, P Salin, J Bullier, and G Horsburgh, *Topography of developing thalamic and cortical pathways in the visual system of the cat*, The Journal of Comparative Neurology **348** (1994), no. 2, 298–319.

[KSM03] Nir Kalisman, Gilad Silberberg, and Henry Markram, *Deriving physical connectivity from neuronal morphology*, Biological Cybernetics **88** (2003), no. 3, 210–218.

[KSM05] ———, *The neocortical microcircuit as a tabula rasa*, Proceedings of the National Academy of Sciences of the United States of America **102** (2005), no. 3, 880–885.

[KY96] E D Kirson and Y Yaari, *Synaptic NMDA receptors in developing mouse hippocampal neurones: functional properties and sensitivity to ifenprodil*, The Journal of Physiology **497 (Pt 2)** (1996), 437–455.

[LF07] Joachim Lübke and Dirk Feldmeyer, *Excitatory signal flow and connectivity in a cortical column: focus on barrel cortex*, Brain Structure and Function **212** (2007), no. 1, 3–17.

[LL10] Debora Ledergerber and Matthew Evan Larkum, *Properties of layer 6 pyramidal neuron apical dendrites.*, The Journal of neuroscience : the official journal of the Society for Neuroscience **30** (2010), no. 39, 13031–13044.

[LRDL06] Mikael Lundqvist, Martin Rehn, Mikael Djurfeldt, and Anders Lansner, *Attractor dynamics in a modular network model of neocortex*, Network (Bristol, England) **17** (2006), no. 3, 253–276.

[LRF99] I Lampl, I Reichova, and D Ferster, *Synchronous membrane potential fluctuations in neurons of the cat visual cortex.*, Neuron **22** (1999), no. 2, 361–374.

[Maa00] W Maass, *On the computational power of winner-take-all*, Neural Computation (2000).

[Maa07] ———, *Liquid Computing*, LECTURE NOTES IN COMPUTER SCIENCE **4497** (2007), 507–516.

[MBCS05] Risto Miikkulainen, JA Bednar, Y Choe, and Joseph Sirosh, *Computational Maps in the Visual Cortex*, Book (2005).

[MBG04] Hamish Meffin, Anthony N Burkitt, and David B Grayden, *An analytical model for the "large, fluctuating synaptic conductance state" typical of neocortical neurons in vivo.*, Journal of Computational Neuroscience **16** (2004), no. 2, 159–175.

[MBLP06] W Ma, J Beck, P Latham, and A Pouget, *Bayesian inference with probabilistic population codes*, Nature Neuroscience (2006).

[MBS02] Troy W Margrie, Michael Brecht, and Bert Sakmann, *In vivo, low-resistance, whole-cell recordings from neurons in the anaesthetized and awake mammalian brain*, Pflügers Archiv : European journal of physiology **444** (2002), no. 4, 491–498.

[MCLP85] D A McCormick, B W Connors, J W Lighthall, and D A Prince, *Comparative electrophysiology of pyramidal and sparsely spiny stellate neurons of the neocortex*, Journal of Neurophysiology **54** (1985), no. 4, 782–806.

[MGM00] S J Martin, P D Grimwood, and R G Morris, *Synaptic plasticity and memory: an evaluation of the hypothesis*, Annual Review of Neuroscience **23** (2000), 649–711.

[MLF+97] H Markram, J Lübke, M Frotscher, A Roth, and B Sakmann, *Physiology and anatomy of synaptic connections between thick tufted pyramidal neurones in the developing rat neocortex*, The Journal of Physiology **500 (Pt 2)** (1997), 409–440.

[MLFS97] H Markram, J Lübke, M Frotscher, and B Sakmann, *Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs.*, Science (New York, NY) **275** (1997), no. 5297, 213–215.

[Mou57] Vernon B. Mountcastle, *Modality and topographic properties of single neurons of cat's somatic sensory cortex.*, Journal of Neurophysiology **20** (1957), no. 4, 408–434.

[MP43] WS McCulloch and W Pitts, *A statistical consequence of the logical calculus of nervous nets*, Bulletin of Mathematical Biology (1943).

[MSTN03] Chaelon Myme, Ken Sugino, Gina Turrigiano, and Sacha Nelson, *The NMDA-to-AMPA Ratio at Synapses Onto Layer 2/3 Pyramidal Neurons Is Conserved Across Prefrontal and Visual Cortices*, Journal of Neurophysiology **90** (2003), no. 2, 771.

[OB11] Srdjan Ostojic and Nicolas Brunel, *From spiking neuron models to linear-nonlinear models.*, PLoS Computational Biology **7** (2011), no. 1, e1001056.

[OF05] Bruno A Olshausen and David J Field, *How close are we to understanding V1?*, Neural Computation **17** (2005), no. 8, 1665–1699.

[OL08] Michael Okun and Ilan Lampl, *Instantaneous correlation of excitation and inhibition during ongoing and sensory-evoked activities.*, Nature Neuroscience **11** (2008), no. 5, 535–537.

[Pav60] I Pavlov, *Conditioned reflexes*, Oxford University Press (1960).

[PBD+99] H Pawelzik, A P Bannister, J Deuchars, M Ilia, and A M Thomson, *Modulation of bistratified cell IPSPs and basket cell IPSPs by pentobarbitone sodium, diazepam and Zn2+: dual recordings in slices of adult rat hippocampus.*, The European journal of neuroscience **11** (1999), no. 10, 3552–3564.

[PBM11] Rodrigo Perin, Thomas K Berger, and Henry Markram, *A synaptic organizing principle for cortical neuronal groups.*, PNAS **108** (2011), no. 13, 5419–5424.

[PDDL99] A Pouget, S Deneve, J Ducom, and P Latham, *Narrow versus wide tuning curves: What's best for a population code?*, Neural Computation (1999).

[Pea88] Judea Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*, San Francisco, CA, USA, 1988.

[PGBZ+06] N V Povysheva, G Gonzalez-Burgos, A V Zaitsev, S Kröner, G Barrionuevo, D A Lewis, and L S Krimer, *Properties of excitatory synaptic responses in fast-spiking interneurons and pyramidal cells from monkey and rat prefrontal cortex*, Cerebral cortex (New York, NY : 1991) **16** (2006), no. 4, 541–552.

[PHT03] H Pawelzik, D I Hughes, and A M Thomson, *Modulation of inhibitory autapses and synapses on rat CA1 interneurones by GABA(A) receptor ligands.*, The Journal of Physiology **546** (2003), no. Pt 3, 701–716.

[PKD+06] D Price, H Kennedy, C Dehay, L Zhou, and M Mercier, *The development of cortical connections*, European Journal of Neuroscience (2006).

[PMBA+09] Frédéric Pouille, Antonia Marin-Burgin, H Adesnik, BV Atallah, and Massimo Scanziani, *Input normalization by global feedforward inhibition expands cortical dynamic range*, Nature Neuroscience **12** (2009), no. 12, 1577–1585.

[PS97] A Pouget and T Sejnowski, *Spatial transformations in the parietal cortex using basis functions*, Journal of Cognitive Neuroscience **9** (1997), no. 2, 222–237.

[RBH+11] Alex Roxin, Nicolas Brunel, David Hansel, Gianluigi Mongillo, and Carl van Vreeswijk, *On the distribution of firing rates in networks of cortical neurons.*, The Journal of neuroscience : the official journal of the Society for Neuroscience **31** (2011), no. 45, 16217–16226.

[RBS+67] W Rall, R E Burke, T G Smith, P G Nelson, and K Frank, *Dendritic location of synapses and possible mechanisms for the monosynaptic EPSP in motoneurons*, Journal of Neurophysiology **30** (1967), no. 5, 1169–1193.

[RD06] M Rudolph and Alain Destexhe, *How much can we trust neural simulation strategies?*, Neurocomputing (2006).

[RD09] U Rutishauser and R Douglas, *State-Dependent Computation Using Coupled Recurrent Networks*, Neural Computation (2009).

[RDS10] Ueli Rutishauser, Rodney J Douglas, and Jean-Jacques Slotine, *Collective Stability of Networks of Winner-Take-All Circuits.*, Neural Computation (2010).

[Rey03] A Reyes, *Synchrony-dependent propagation of firing rate in iteratively constructed networks in vitro*, Nature Neuroscience (2003).

[RGD+11] Alexander Rast, Francesco Galluppi, Sergio Davies, Luis Plana, Cameron Patterson, Thomas Sharp, David Lester, and Steve Furber, *Concurrent heterogeneous neural model simulation on real-time neuromimetic hardware.*, Neural networks : the official journal of the International Neural Network Society **24** (2011), no. 9, 961–978.

[RHK+11] Srikanth Ramaswamy, Sean Lewis Hill, James Gonzalo King, Felix Schürmann, Yun Wang, and Henry Markram, *Intrinsic Morphological Diversity of Thick-tufted Layer 5 Pyramidal Neurons Ensures Robust and Invariant Properties of in silico Synaptic Connections.*, The Journal of Physiology (2011).

[RHW86] D E Rumelhart, GE Hinton, and R J Williams, *Learning representations by back-propagating errors*, Nature **323** (1986), 533–536.

[Ric77] LM Ricciardi, *Diffusion processes and related topics in biology*, Springer-Verlag, 1977.

[RLCL+03] Alexander Rauch, Giancarlo La Camera, Hans-Rudolf Luscher, Walter Senn, and Stefano Fusi, *Neocortical pyramidal cells respond as integrate-and-fire neurons to in vivo-like input currents.*, Journal of Neurophysiology **90** (2003), no. 3, 1598–1612.

[RMAH07] Victor M Rodriguez-Molina, Ad Aertsen, and Detlef H Heck, *Spike Timing and Reliability in Cortical Pyramidal Neurons: Effects of*

EPSC Kinetics, Input Synchronization and Background Noise on Spike Timing, PloS one **2** (2007), no. 3, e319.

[RMV01] D S Reich, F Mechler, and J D Victor, *Independent and redundant information in nearby cortical neurons.*, Science (New York, NY) **294** (2001), no. 5551, 2566–2568.

[Ros58] F Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological Review **65** (1958), no. 6, 386–408.

[RP99] M Riesenhuber and T Poggio, *Hierarchical models of object recognition in cortex*, Nature Neuroscience (1999).

[RRMS10] U Rutishauser, IB Ross, AN Mamelak, and EM Schuman, *Human memory strength is predicted by theta-frequency phase-locking of single neurons*, Nature (2010), online first.

[RRWF10] Mattia Rigotti, Daniel Ben Dayan Rubin, Xiao-Jing Wang, and Stefano Fusi, *Internal representation of task rules by recurrent dynamics: the importance of the diversity of neural responses.*, Frontiers in computational neuroscience **4** (2010), 24.

[RW72] R Rescorla and A Wagner, *Variations in the Effectiveness of Reinforcement and Nonreinforcement*, Classical Conditioning. II: Current Research and Theory (1972), 64–99.

[SA01] E Salinas and LF Abbott, *Coordinate transformations in the visual system: how to generate gain fields and what to compute with them*, Progress in brain research **130** (2001), 175–190.

[Sch78] C N Scholfield, *Electrical properties of neurones in the olfactory cortex slice in vitro*, The Journal of Physiology **275** (1978), 535–546.

[Sie51] AJF Siegert, *Phys. Rev. 81, 617 (1951): On the First Passage Time Probability Problem*, Physical Review (1951).

[SK93] W R Softky and C Koch, *The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs*, The Journal of neuroscience : the official journal of the Society for Neuroscience **13** (1993), no. 1, 334–350.

[SK08] Matthew A Smith and Adam Kohn, *Spatial and temporal scales of neuronal correlation in primary visual cortex.*, The Journal of neuroscience : the official journal of the Society for Neuroscience **28** (2008), no. 48, 12591–12603.

[SKD95] H Suarez, C Koch, and R Douglas, *Modeling Direction Selectivity of Simple Cells in Striate Visual Cortex Within the Framework of the Canonical Microcircuit*, Journal of Neuroscience (1995).

[SM10] Sylvia Schröder and Kevan Martin, *Heterogeneity in responses of neighboring neurons in cat primary visual cortex*, AREADNE (Santorini, Greece), Institute of Neuroinformatics, University of Zurich & ETH Zurich, Switzerland, June 2010, p. 89.

[SS95] G Stuart and B Sakmann, *Amplification of EPSPs by axosomatic sodium channels in neocortical pyramidal neurons*, Neuron **15** (1995), no. 5, 1065–1076.

[SS01] E Salinas and T Sejnowski, *Gain modulation in the central nervous system: where behavior, neurophysiology, and computation meet*, The Neuroscientist (2001), 430–440.

[STHM+96] K J Stratford, K Tarczy-Hornoch, K A Martin, N J Bannister, and J J Jack, *Excitatory synaptic inputs to spiny stellate cells in cat visual cortex.*, Nature **382** (1996), no. 6588, 258–261.

[TD94] A M Thomson and J Deuchars, *Temporal and spatial properties of local circuits in neocortex*, Trends in Neurosciences **17** (1994), no. 3, 119–126.

[TDW93a] A M Thomson, J Deuchars, and D C West, *Large, deep layer pyramid-pyramid single axon EPSPs in slices of rat motor cortex display paired pulse and frequency-dependent depression, mediated presynaptically and self-facilitation, mediated postsynaptically*, Journal of Neurophysiology **70** (1993), no. 6, 2354–2369.

[TDW93b] AM Thomson, J Deuchars, and DC West, *Single axon excitatory postsynaptic potentials in neocortical interneurons exhibit pronounced paired pulse facilitation*, Neuroscience **54** (1993), no. 2, 347–360.

[THMJS98] K Tarczy-Hornoch, K A Martin, J J Jack, and K J Stratford, *Synaptic interactions between smooth and spiny neurones in layer 4 of cat visual cortex in vitro.*, The Journal of Physiology **508 (Pt 2)** (1998), 351–363.

[THMSJ99] K Tarczy-Hornoch, K A Martin, K J Stratford, and J J Jack, *Intracortical excitation of spiny neurons in layer 4 of cat striate cortex in vitro.*, Cerebral cortex (New York, NY : 1991) **9** (1999), no. 8, 833–843.

[TN00] G G Turrigiano and S B Nelson, *Hebb and homeostasis in neuronal plasticity*, Current Opinion in Neurobiology **10** (2000), no. 3, 358–364.

[TN04] G Turrigiano and S Nelson, *Homeostatic plasticity in the developing nervous system*, Nature Reviews Neuroscience **5** (2004), 97–107.

[Tre03] A Treves, *Computational Constraints that may have Favoured the Lamination of Sensory Cortex*, Journal of Computational Neuroscience (2003).

[TSB98] G Tamás, P Somogyi, and E H Buhl, *Differentially interconnected networks of GABAergic interneurons in the visual cortex of the cat.*, The Journal of neuroscience : the official journal of the Society for Neuroscience **18** (1998), no. 11, 4255–4270.

[Tuc88] Henry C. Tuckwell, *Introduction to theoretical neurobiology*, Cambridge studies in mathematical biology, 8, Cambridge University Press, April 1988.

[Tur08] Gina G Turrigiano, *The self-tuning neuron: synaptic scaling of excitatory synapses*, Cell **135** (2008), no. 3, 422–435.

[TW93] A M Thomson and D C West, *Fluctuations in pyramid-pyramid excitatory postsynaptic potentials modified by presynaptic firing pattern and postsynaptic membrane potential using paired intracellular recordings in rat neocortex*, Neuroscience **54** (1993), no. 2, 329–346.

[TWWB02] Alex M Thomson, David C West, Yun Wang, and A Peter Bannister, *Synaptic connections and small circuits involving excitatory and inhibitory neurons in layers 2-5 of adult rat and cat neocortex: triple intracellular recordings and biocytin labelling in vitro*, Cerebral cortex (New York, NY : 1991) **12** (2002), no. 9, 936–953.

[VA05] Tim P Vogels and L F Abbott, *Signal propagation and logic gating in networks of integrate-and-fire neurons.*, The Journal of neuroscience :

the official journal of the Society for Neuroscience **25** (2005), no. 46, 10786–10795.

[VA09] ———, *Gating multiple signals through detailed balance of excitation and inhibition in spiking networks*, Nature Neuroscience **12** (2009), no. 4, 483–491.

[VJSK11] Zsuzsanna Varga, Hongbo Jia, Bert Sakmann, and Arthur Konnerth, *Dendritic coding of multiple sensory inputs in single cortical neurons in vivo.*, PNAS **108** (2011), no. 37, 15420–15425.

[Wan99] X J Wang, *Synaptic basis of cortical persistent activity: the importance of NMDA receptors to working memory*, The Journal of neuroscience : the official journal of the Society for Neuroscience **19** (1999), no. 21, 9587–9603.

[WGTR+02] Yun Wang, Anirudh Gupta, Maria Toledo-Rodriguez, Cai Zhi Wu, and Henry Markram, *Anatomical, physiological, molecular and circuit properties of nest basket cells in the developing somatosensory cortex*, Cerebral cortex (New York, NY : 1991) **12** (2002), no. 4, 395–410.

[WJW+02] KK Waikul, L Jiang, EC Wilson, FC Harris, and PH Goodman, *Design and implementation of a web portal for a NeoCortical Simulator*, Proceedings of the 17th International Conference on Computers and their Applications (CATA 2002), 2002, pp. 349–353.

[WL90] B Widrow and M A Lehr, *30 years of adaptive neural networks: perceptron, Madaline, and backpropagation*, Proceedings of the IEEE **78** (1990), no. 9, 1415–1442.

[WLHP68] A R Wagner, F A Logan, K Haberlandt, and T Price, *Stimulus selection in animal discrimination learning*, Journal of experimental psychology **76** (1968), no. 2, 171–180.

[WSFS10] H P Wang, D Spencer, J M Fellous, and T J Sejnowski, *Synchrony of Thalamocortical Inputs Maximizes Cortical Reliability*, Science (New York, NY) **328** (2010), no. 5974, 106–109.

[WSS05] Jack Waters, Andreas Schaefer, and Bert Sakmann, *Backpropagating action potentials in neurones: measurement, mechanisms and potential functions*, Progress in Biophysics and Molecular Biology **87** (2005), no. 1, 145–170.

[WvdM76] D J Willshaw and C von der Malsburg, *How Patterned Neural Connections Can Be Set Up by Self-Organization*, Proceedings of the Royal Society of London. Series B **194** (1976), no. 1117, 431–445.

[WvRM+00] A J Watt, M C van Rossum, K M MacLeod, S B Nelson, and G G Turrigiano, *Activity coregulates quantal AMPA and NMDA currents at neocortical synapses*, Neuron **26** (2000), no. 3, 659–670.

[WW07] C Weber and S Wermter, *A self-organizing map of sigma–pi units*, Neurocomputing (2007).

[YFW05] J Yedidia, W Freeman, and Y Weiss, *Constructing free-energy approximations and generalized belief propagation algorithms*, Information Theory (2005).

[YKA89] WM Yamada, C Koch, and PR Adams, *Multiple channels and calcium dynamics*, Methods in neuronal modeling: From synapses to networks (C Koch and I Segev, eds.), MIT Press, 1989.

[ZA88] D Zipser and RA Andersen, *A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons*, Nature **331** (1988), no. 6158, 679–684.

[ZR93] D Zipser and D Rumelhart, *The neurobiological significance of the new learning models*, Computational neuroscience (1993), 192–200.

Curriculum Vitae

Florian Jug

born on April 8, 1979
citizen of Austria

1985 - 1989	Volksschule in Carinthia, Austria
1989 - 1992	Hauptschule in St. Paul, Carinthia, Austria
1992 - 1994	Realschule in Ismaning, Bavaria, Germany
1994 - 1999	HTL for Information Technology and Management, Villach, Carinthia, Austria
1996	Internship for KET and BMW, Munich, Germany
1998	Internship for KET and BMW, Munich, Germany
1999 - 2005	Studies of Computer Science at TU Munich, Germany – Degree: Dipl. Inf. TU
1999 - 2005	Employee at S+W CA Technik GmbH, Munich, Germany – Position: leading developer in several SW-development projects for BMW and EDAG.
since 2005	Ph.D. student at ETH Zurich, Switzerland