

Probabilistic neural computation and neural simulation technology

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Master of Science

Physik

Jakob Jordan

aus

Bad Driburg

Berichter: Universitätsprofessor Dr. Markus Diesmann
Universitätsprofessor Dr. Bastian Leibe

Tag der mündlichen Prüfung: 30.10.2018

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

“The astonishing hypothesis was simply that the mind is the creation of the cells in the brain. There is nothing else, no magic, no special sauce, only neurons and a dance of information.”

Jeff Hawkins, *On Intelligence*, 2004

Summary

Deciphering the working principles of brain function is of major importance from at least two perspectives. From the clinical viewpoint, a deeper understanding of our brains will lead to better treatments for psychological and neurodegenerative diseases. The technological perspective promises smart machines that rival our ability to perceive, learn and act in the real world. It is generally believed that the relevant physical processes can be understood in terms of large, plastic networks of nerve cells. Over the last decade, probability theory has gained popularity as a normative model of brain function, since it offers a unifying view for many behavioural phenomena. To connect this high-level description to low-level implementations in neural substrates, computational models provide effective means to investigate specific hypotheses. Many neural network models for probabilistic inference employ approximation algorithms, relying on stochasticity to achieve a sample-based representation of probability distributions.

In the first part of this thesis, we investigate biophysically plausible sources of stochasticity for models of approximate inference in neural substrates. We consider network models derived from a popular model class for sample-based inference, Boltzmann machines, but equipped with a natural source of stochasticity: synaptic input from other neurons. We demonstrate that stochasticity generated by recurrent neural networks can outperform other approaches based on independent random number streams if resources are limited. We further show that the essential effect required for this approach is also present in networks emulated in a mixed-signal neuromorphic system with strong temporal and spatial heterogeneities. Subsequently we consider network models with alternative sources of stochasticity, including variability in the coupling strength between neurons. Our results show that while these networks are able to represent well-defined probability distributions, probabilistic inference cannot be implemented using straightforward methods familiar from Boltzmann machines.

The human brain contains approximately one hundred billion individual nerve cells connected by about one hundred trillion synapses: This complexity poses a significant computational challenge for the numerical simulation of such systems. To support simulations of large-scale network models at cellular resolution, researchers can rely on well-tested high-performance simulation software that makes efficient use of the largest supercomputers available today with tens of thousands of compute nodes. However, it has an inherent scalability bottleneck arising from the memory consumption of the connection infrastructure and the undirected communication algorithms typically employed.

In the second part of this thesis, we introduce a new two-tier connection infrastructure employing directed communication across compute nodes. By implementing this new technology in NEST, a widely-used simulator for large-scale neural network models, we demonstrate that it solves previous scalability bottlenecks in terms of memory consumption. In addition, we show that on current supercomputers, the directed communication between processes significantly improves the simulation performance, and at the same time maintains high efficiency in small-scale simulations executed on laptops and small clusters.

Simulation software is however only one part required for neuroscientific modeling: for functional networks, providing a realistic environment that interacts with the models is equally important. Researchers should not have to rely on handcrafted solutions as this makes models difficult to evaluate and compare objectively. To tackle these issues, we develop a toolchain that enables closed-loop interactions between neural-network models in established simulators and reinforcement-learning environments from machine-learning toolkits.

In conclusion, this thesis brings models for approximate inference in neural circuits further into the biophysically plausible domain, with direct applications for neuromorphic implementations, and presents new simulation technology preparing software tools for the simulation of human cortex at the resolution of individual neurons and synapses on future HPC systems.

Zusammenfassung

Die Funktionsweise unseres Gehirns zu entschlüsseln ist aus mindestens zwei Perspektiven von dringender Wichtigkeit. Aus klinischer Sicht wird ein besseres Verständnis des Gehirns zu besseren Therapiemöglichkeiten für psychische und neurodegenerative Erkrankungen führen. Die technische Perspektive verspricht schlaue Maschinen, die wir die Welt wie wir wahrnehmen, daraus lernen und entsprechend handeln können. Im Allgemeinen wird angenommen, dass die relevanten physikalischen Prozesse die unserem Gehirn zugrunde liegen, auf der Ebene von großen, plastischen Netzwerken von Nervenzellen verstanden werden können. Über die letzten Dekade hat sich die Wahrscheinlichkeitstheorie als normatives Modell von Hirnfunktion großer Beliebtheit erfreut, da sie eine einheitliche Sichtweise auf das Verhalten von intelligenten Organismen ermöglicht. Computermodelle bieten hervorragende Mittel um diese abstrakte Beschreibungsebene mit den zugrunde liegenden neuronalen Netzwerken zu verbinden. Vielen Netzwerkmodelle zur neuronalen Implementation von wahrscheinlichkeitstheoretischen Berechnungen, liegen approximative Algorithmen zugrunde. Diese benutzen Stochastizität um eine sample-basierte Repräsentation von Wahrscheinlichkeitsverteilungen zu erreichen.

In dem ersten Teil dieser Arbeit untersuchen wir biophysikalisch plausible Quellen von Stochastizität in neuronalen Substraten. Hierfür benutzen wir ein Netzwerkmodell, das von einer weit verbreiteten Klasse für approximative Algorithmen, so genannten Boltzmann Maschinen, abgeleitet wurde. In diesem ersetzen wir die sonst angenommene Stochastizität durch eine natürliche Alternative: Input von anderen Neuronen. Wir zeigen, dass Stochastizität die durch ein zusätzliches rekurrentes Netzwerk von Neuronen generiert wird, besser geeignet ist als die von unabhängigen Zufallszahlengeneratoren, falls Ressourcen limitiert sind. Daraufhin zeigen wir dass der grundlegende Effekt der hierfür zuständig ist, auch in Netzwerken, die in neuromorphen Systemen mit starken räumlichen und zeitlichen Heterogenitäten emuliert werden, vorhanden ist. Wir untersuchen zusätzlich alternative Quellen von Stochastizität, unter Anderem Variabilität in der Kopplungsstärke zwischen Neuronen. Unsere Ergebnisse zeigen, dass Netzwerke die von der Ursprünglichen Definition von Boltzmann Maschinen abweichen, zwar wohldefinierte Wahrscheinlichkeitsverteilungen repräsentieren können, aber dass sie wichtige Eigenschaften verlieren, die Berechnungen in Boltzmann Maschinen besonders effizient machen.

Das Menschliche Gehirn enthält in etwa einhundert Milliarden individuelle Nervenzellen, die über ungefähr einhundert Billionen Synapsen miteinander verschaltet sind. Diese Komplexität stellt Computersimulationen vor enorme Herausforderungen. Um große Netzwerke mit einer Auflösung von einzelnen Neuronen und Synapsen simulieren zu können, wird hoch performante und gut getestete Simulationssoftware benutzt die die Leistung von Supercomputer effizient ausnutzen kann. Moderne Software skaliert bis zu den größten Systemen die momentan verfügbar sind, in denen zehntausende von Rechenknoten eingesetzt werden. Die momentane neuronale Simulationstechnologie hat jedoch einen Skalierbarkeitseingpass, der sich auf die verwendeten Verbindungsinfrastrukturen und Kommunikationssalgorithmen zurückführen lässt.

In dem zweiten Teil dieser Arbeit präsentieren wir eine neue, zweigeteilte Verbindungsinfrastruktur, die gerichtete Kommunikation zwischen verschiedenen Prozessen ermöglicht. Wir implementieren diese neue Technologie in NEST, einem weit verbreiteten Simulator für großskalige neuronale Netzwerkmodelle, und zeigen, dass diese Technologie die vorherigen Skalierbarkeitseingpässe entfernt. Unsere Ergebnisse demonstrieren zusätzlich, dass diese Änderungen Simulationen auch auf aktuellen Supercomputern signifikant schneller machen, während die Leistung von Simulationen auf kleinen Systemen nicht negativ beeinflusst wird.

Zusätzlich zu Software für die Simulation von neuronalen Netzwerken, ist auch Software die diese Netzwerke mit realistischen Input versorgt, notwendig, um beispielsweise Netzwerkmodelle von Lernen zu untersuchen. Hierbei sollte nicht speziell für ein bestimmte Studie entwickelte Software zum Einsatz kommen, da dies die Vergleichbarkeit von Modellen zusätzlich erschwert. Um dieses Problem zu vermeiden, entwickeln wir eine Toolchain die eine enge Interaktion zwischen neuronalen Netzwerkmodellen in weit verbreiteten Simulatoren und standardisierten simulierten Umgebungen ermöglicht.

Zusammenfassend betrachtet bringt diese Arbeit neuronale Netzwerkmodelle die Wahrscheinlichkeitsberechnungen implementieren näher zusammen mit biophysikalischen Erkenntnissen, und stellt Simulationstechnologie vor, die Software auf die Simulation des menschlichen Kortex mit einer Auflösung von einzelnen Nervenzellen und Synapsen auf der nächsten Generation von Supercomputern vorbereitet.

Contents

Summary	v
Zusammenfassung	vii
 I Fundamentals	 1
1 The probabilistic brain	3
1.1 Plausible reasoning	3
1.2 Approximate inference	8
1.3 Markov-chain-Monte-Carlo methods	10
2 Computational neuroscience	15
2.1 Defining the microscopic level	19
2.2 Single neurons	20
2.2.1 Biophysics	20
2.2.2 A mathematical description of neurons and synapses	24
2.3 Bottom-up network models	29
2.4 Top-down network models	31
2.5 Probabilistic computations in neural substrates	32
3 Simulation technology	39
3.1 Computational challenges of large-scale neural-network simulations	41
3.2 Software tools	42
3.3 Neuromorphic hardware	44
4 Scope and structure of this thesis	47
 II Results	 51
5 Stochastic neural computation without noise	53
5.1 Introduction	54
5.2 Methods	55
5.2.1 Binary network simulation	55
5.2.2 Random sampling networks	56
5.2.3 Intrinsic noise	56
5.2.4 Private noise	57
5.2.5 Shared noise	59
5.2.6 Network noise	61
5.2.7 Calibration (binary networks)	63
5.2.8 Handwritten-digit generation	63
5.2.9 Calibration (spiking networks)	64
5.3 Results	65

5.3.1	Networks with private Gaussian noise approximate Boltzmann machines	65
5.3.2	Shared-noise correlations impair sampling performance	66
5.3.3	Network-generated noise recovers sampling performance	67
5.3.4	Deterministic recurrent neural networks as a noise source for a generative model of handwritten digits	68
5.3.5	Shared-input correlations impair network performance for high-entropy tasks	68
5.3.6	Small recurrent networks provide large sampling networks with noise	69
5.3.7	Networks of spiking neurons implement neural sampling without noise	70
5.4	Discussion	71
6	The effect of heterogeneity on decorrelation mechanisms	75
6.1	Introduction	76
6.2	Methods	79
6.2.1	Network model	79
6.2.2	Network emulations on the neuromorphic-hardware system <i>Spikey</i>	79
6.2.3	Experimental setup	81
6.2.4	Reproducibility of hardware emulations	82
6.2.5	Calibration	82
6.2.6	Correlation measures	84
6.3	Results	85
6.3.1	Decorrelation by inhibitory feedback	85
6.3.2	Effect of heterogeneity on decorrelation	89
6.4	Discussion	91
7	Approximate inference in non-Boltzmann networks	97
7.1	Fast inference via constraints in Boltzmann machines	97
7.2	Computing the stationary distribution of arbitrary sampling networks	99
7.3	Sampling networks with Gaussian noise	103
7.4	The role of synaptic variability in neural computation	104
7.5	Sampling in asymmetric networks	109
7.6	Discussion	110
8	Extremely scalable spiking neural network simulation code	113
8.1	Introduction	114
8.2	Materials and Methods	117
8.2.1	Benchmark network model	117
8.2.2	Measuring scalability	117
8.2.3	Supercomputers	118
8.2.4	NEST simulator	119
8.2.5	Adaptation of memory-usage model	120
8.3	Results	121
8.3.1	Two-tier connection infrastructure	122
8.3.2	Communication of spikes	128
8.3.3	Adaptations for the pre-petascale regime	130
8.3.4	Memory usage of the new simulation kernel	130
8.3.5	Performance of new simulation kernel	131

8.4	Detailed profiling via HPCToolkit	137
8.5	Discussion	140
9	Interfacing neural network simulators and the OpenAI Gym	147
9.1	Introduction	148
9.2	Pre-existing components	150
9.3	Results	150
9.3.1	Extending the ROS - MUSIC toolchain	150
9.3.2	ZeroMQ wrapper for the OpenAI Gym	151
9.3.3	Applications	152
9.4	Derivation of the learning rule	156
9.5	Conclusion	158
III	Conclusion & Outlook	161
10	Conclusion	163
11	Outlook	167
IV	Appendices	171
A	Supplementary material to Chapter 5	173
A.1	Pairwise input correlations	173
A.2	Sampling error depends on number of noise inputs per sampling unit	174
A.3	Small, recurrent networks can supply large sampling networks with noise – no weight scaling	174
A.4	Simulation details	175
B	Supplementary material to Chapter 6	185
B.1	Network description	185
B.2	Description of data analysis	185
B.3	Power consumption	185
B.4	Modification of the bisection method	189
B.5	Effective weights	189
B.6	Simulations with software	190
B.7	Linear model	193
B.8	Firing statistics in the RAND case	195
B.9	Correlation matrices	196
B.10	Results for different <i>Spikey</i> chips	197
B.11	Reproducibility of networks with intact feedback	197
B.12	Extraction of effective PSP amplitudes, time constants and delays . . .	200
C	Supplementary material to Chapter 8	207
C.1	Network and simulation parameters	207
C.2	Connection infrastructure for devices	207
C.3	Scalability of <code>MPI_Allgather</code> and <code>MPI_Alltoall</code>	211
C.4	Structural plasticity	212
C.5	Strong scaling over threads	212
C.6	Strong scaling over MPI processes (full data)	213

D Supplementary material to Chapter 9	215
D.1 Network description	215
D.2 JSON Message types	215
D.3 Example wrapper configuration file	215
D.4 Example MUSIC configuration file	220
D.5 Environments	220
Bibliography	223
Author's list of publications	249
Acknowledgements	251
Funding	253

Für meine Eltern und Oma

Part I

Fundamentals

Chapter 1

The probabilistic brain

[T]he probability calculus is a formalization and abstraction of the way we do uncertain reasoning. [...] So “of course” the brain “implements” a basic form of Bayes’ theorem: that’s a constation, not something to prove. Probability theory starts from there.

P.G.L. Porta Mana, pers. comm., 2017

1.1 Plausible reasoning

Humans and other intelligent animals are remarkable adept at making sense of a never ending stream of uncertain and ambiguous perceptual information. We routinely draw conclusions from incomplete and noisy observations about properties of the world and act accordingly. We are able to learn from our experiences and make predictions about future events. Even if we have never experienced a certain situation before, we can reasonably construct possible futures based on previous knowledge – a feat that is already present in one-year-old infants (Téglás et al., 2011). In all these tasks the ability to represent and consider uncertainty is of tremendous importance – failure to account for this will inevitable entail mildly inconvenient to disastrous results.

Usually we are able to consider the omnipresent uncertainty, even without consciously thinking about it: Should I bring an umbrella or sunscreen? Can I safely cross the street? Who took my lunch? None of these questions can be answered with 100% certainty, nevertheless we address them effortlessly. Imagine you have a flight to catch and the only time available is from your mechanical wristwatch. Of course you want to leave early enough to make to not miss your flight. If you have just set your watch according to a radio clock, you are very certain about the time it displays, so you leave when it shows that in 2h your plane is scheduled to take off. However, imagine just after setting your watch you see your child shortly fiddling with the watch’s crown. You were all the way across the room, and consequently do not know how much this has altered the displayed time, and in particular whether the watch is now running slow or fast. Under these circumstances you might be inclined to leave your house earlier than in the case of an accurate watch, because the uncertainty about the correct time has significantly increased. The capacity of humans to carry out this more-or-less accurate reasoning with considerable ease and take appropriate action even in the presence of uncertainty is usually referred to as common sense – and yours will most likely be doubted if you rely on an inaccurate watch as much as on an accurate one. Furthermore, if several people are asked

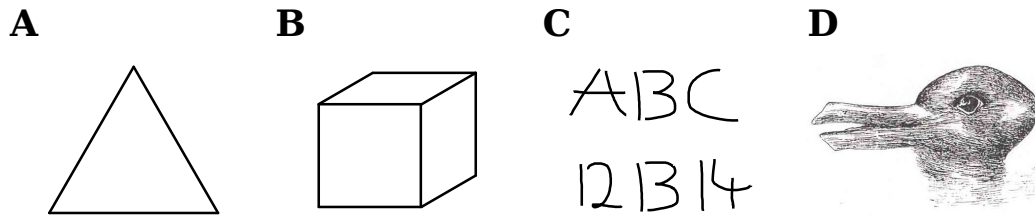


FIGURE 1.1: **A:** Simple stimulus. **B:** Stimulus leading to inference of hidden cause from experience. **C:** Stimulus leading to inference of hidden cause from context, after Kahneman (2011). **D:** Ambiguous stimulus giving rise to perceptual bistability (Jastrow, 1899).

to reason about the circumstances described above, they generally tend to come to the same conclusions within some variation. Considering the enormous amount of uncertainty we face in certain situations, this consistency of reasoning between individuals should not be regarded as coincidental, but hinting at a fundamental, structured way of handling uncertain statements.

Similar types of intuitive judgment are involved already at the level of unconscious perception: consider reconstructing a $3d$ object from the two $2d$ projections on your retina, ignoring for a moment all physical sources of uncertainty, like the limited spatial resolution of the retina or photon shot-noise. Logically, this reconstruction is impossible since the problem is severely underdetermined and an infinite number of possible $3d$ shapes can give rise to the same $2d$ projections. In order to arrive at a useful estimate of the identity of the $3d$ object, we hence need to include previous experience about the most likely cause for the observed $2d$ projections (Von Helmholtz, 1867; Knill and Richards, 1996). However, even taking into account this previous knowledge, our final guess is not necessarily stable as demonstrated by various optical illusions. We perform these types of guesses with incredible ease, with so much ease in fact, that it is sometime difficult to see exactly what is actually there. Have a look at figure 1.1, panel **A**. Most likely you will see three black lines forming a triangle. Now consider panel **B**: do you see a square and two parallelograms? No, most likely you interpret the combination of lines as stemming from the two-dimensional depiction of a three-dimensional cube, and it is difficult so see it in the former interpretation. In addition, our perception depends on context. Read the letters and digits in panel **C**. Did you notice how the middle part of the two lines is identical? Independent of whether you did, you probably did not read the first line as A13C or the second as 12B14. Panel **D** finally demonstrates nicely that even if you have arrived at a definite conclusion about what you see, it is not so clear cut after all. Maybe you see a duck, but seconds later you might see a rabbit.

The thought process underlying this type of reasoning cannot be deductive following the rules of traditional logic since you cannot know anything about the future (or the past, or the present for that matter) with certainty. The whole process must rather be inductive in nature: You perform inferences from particular, uncertain observations to general “real” features of the world. Nevertheless there apparently is a consistent, structured way of reasoning that allows you do carry out plausible inferences. This must allow someone else, or future you, to come to the same conclusion given the same information, even if the specific line of argumentation might be

different.¹ In order to accommodate this inductive process, we hence need to consider an extended form of logic, that can handle uncertain statements rather than imposing the restriction to true or false.

Probability theory, as developed by Ramsey, Cox, Good, Jaynes, Lindley and others provides a mathematical theory of reasoning under uncertainty based on very few axioms about how probabilities combine and relate to each other in a consistent fashion (sum rule, product rule and convexity) (Ramsey, 1950; Cox, 1946; Good, 1950; Jaynes, 2003a; Lindley, 2014). Jaynes (2003a), for example, obtains the axioms starting from the following three desiderata:

1. Representation of degrees of plausibility by real numbers
2. Qualitative correspondence with common sense
3. Consistency

The consistency requirement includes the intuitive condition that if one can reason about a statement in more than one way, every way has to lead to the same conclusion eventually. The formal manipulation of uncertain statements following the laws of probability leads to plausible reasoning. In particular, probability theory formalizes how to make inferences from observations: Given the observations I , the probability of event X is determined by the conditional probability $p(X|I)$. We refer to this process as *probabilistic inference*, although we often just say *inference* for short. We mention in passing that we view probabilities as a degrees of reasonable belief (Jeffreys, 1973) rather than a measure of frequency of occurrence of an observation in a hypothetical infinitely repeatable experiment. For more details, the formal rules and answers to critiques of this view we refer the reader to the classical works (Good, 1950; Jeffreys, 1973; Finetti, 1990a; Finetti, 1990b; Bernardo and Smith, 2000; Jaynes, 2003a; Lindley, 2014).

We would like to emphasize at this point that there is only a single set of rules for reasoning under uncertainty satisfying the above premises and that this set of rules is identical to the laws of probability theory (Jaynes, 1988; Jaynes, 2003a). As Laplace (1820) already pointed out:

“We see, from this Essay, that the theory of probability at bottom is nothing but common sense reduced to calculation. It makes us appreciate with exactness what the right mind feels with a sort of instinct, often without being able to account for it.”

We hence argue that “the right mind” constantly manipulates internal probabilistic models according to the rules of probability theory. This allows us to make the most of the restricted amount of information about the world available through our limited senses. In the current neuroscientific literature, this view is expressed in the “Bayesian coding hypothesis”: to make sense out of the uncertain, ambiguous and incomplete perceptual input, our brain represents and combines information about the world around us using probability distributions (Knill and Richards, 1996; Knill and Pouget, 2004; Chater, Tenenbaum, and Yuille, 2006; Doya, 2007; Fiser et al., 2010; Griffiths et al., 2010; Ma and Jazayeri, 2014; Sanborn and Chater, 2016). Sensory input for example, can be combined with prior knowledge obtained from past experience or evolutionary pressure generating a distribution over possible causes of this sensory stimulation, from which further consequences, e.g., appropriate actions, can be derived. Computing and representing a distribution over possible causes allows

¹However, diverging views can arise when subjects hold widely differing prior information (Jaynes, 2003a, ch. 5.3).

us to represent all possible explanations with their relative probabilities instead of just estimating the most likely interpretation, which by definition would ignore how sure we are about our conclusion. As a formalization of common sense, probability theory is accordingly an appropriate tool to construct mathematical and computational models that describe cognitive processes of humans and other intelligent animals, for example how they perceive the world around them (Olshausen and Field, 1997; Kersten, Mamassian, and Yuille, 2004), perform coherent integration of prior knowledge and learn from previous experience (Kording and Wolpert, 2004; Denison et al., 2013; Haefner, Berkes, and Fiser, 2016) or optimally integrate information across different sensory modalities with different uncertainties (Körding et al., 2007; Morgan, DeAngelis, and Angelaki, 2008; Angelaki, Gu, and DeAngelis, 2009). By making predictions that can be compared to experimental data, these models allow us to pick out certain types of probabilistic models that might actually be employed in intelligent organisms out of the infinite variety offered by the formal theory.

At this point, we need to address an often encountered critique of this view. While in many cases humans indeed behave optimally, i.e., behave as if² they manipulate probabilistic models according to the formal rules, (Kording and Wolpert, 2004; Knill and Pouget, 2004; Ma et al., 2006; Griffiths and Tenenbaum, 2006; Fiser et al., 2010; Denison et al., 2013), it has been repeatedly shown that this is not the case in general. Under particular circumstances systematic biases and “errors of judgment” arise (see, e.g., Tversky and Kahneman, 1973; Kahneman, Slovic, and Tversky, 1982; Tversky and Kahneman, 1985) and the reader can most likely think of many situations in which human reasoning seems to be inconsistent or plain wrong. We hence need to stress that the Bayesian coding hypothesis does not mean that all results obtained from manipulating certain initial statements according to the formal rules are identical to the behavior of humans or other animals (see also Ma, 2012). Rather, the formal calculus is a prescriptive model, providing a normative framework of how intelligent organisms should act in the presence of uncertainty to avoid any inconsistencies with common sense – it is an ideal to which actual human behavior can and should be compared (cf. Lindley, 2014). As with any scientific theory that is concerned with natural phenomena, it describes an idealized reality: the perfectly reasonable human with infinite memory and infinite computational power. It provides a useful abstraction which allows us to understand nature and from which we need to work out predictions that can be compared to the real world – just as much as the ideal gas or point particles in physics or various geometrical forms. Real humans will necessarily demonstrate only “approximate rationality” due to limited memory and limited computational capacity.

There are many ways to reconcile errors of human judgment with the Bayesian coding hypothesis: cognitive biases could be attributed to certain deeply ingrained priors that were evolutionary useful (as in risk-aversion/risk-seeking asymmetries; Tversky and Kahneman, 1992) but are possibly ignored in formal derivations. Additionally they could result from correct application of the formal rules in human thought, but using “wrong parameters” when applying an internal probabilistic model to a new domain (Griffiths and Tenenbaum, 2006), from heuristics that save “computation” time (Gershman, Horvitz, and Tenenbaum, 2015) or from simplifying assumptions about relevant information. Consider the following example.

² “[...] These justifications show that, one the assumption of certain compelling desiderata, a rational person will hold beliefs, and preferences, *as if* he had a system of subjective probabilities and utilities satisfying a familiar set of axioms. He might as well be explicit about it: after all it is doubtful whether any of our knowledge is better than of the ‘as if’ variety.” (Good, 1983)

Which of these two sequences is more likely to result from throwing a fair coin five times (H = heads, T = tails)?

H H H H H
H T H H T

Your intuition might have been that the second one is more likely to occur, although after some active consideration you will agree that the probabilities for each of these sequences to occur exactly like prescribed above, are identical ($1/2^5$). The initial guess might just be the result from a heuristic, where we use the total number of heads and tails to summarize the data and compare these statistics to the expected values of head and tail counts.

From the limitation of finite memory and processing power rather crude inconsistencies in human reasoning may arise. Since every change in your “body of beliefs” (Good, 1950), i.e., internal probabilistic model, consumes a certain amount of time and energy, it is impossible to update every belief you currently hold whenever you gain any new evidence. In some cases, new evidence can be quickly integrated, while in others additional thought is required. In particular keeping your beliefs consistent requires effort and an open mind – “When my information changes, I change my mind. What do you do?”.³ In some cases, probabilities can be changing by “pure thought without any change in the empirical data” (Good, 1983), just as it possibly happened with your judgment about the more likely sequence above. These “dynamic probabilities” (Good, 1983) can arise from a finite speed of information processing in physical systems and can, before converging to a stationary state, differ from the results of the formal theory.

The fact that all physical systems have limited computational power means that exact probabilistic inference is computationally intractable for any real-world tasks involving many random variables. Any system that needs to take into account the costs of the inference procedure itself and costs of collecting data (leading to “type-2 rationality”, Good (e.g., 1983) and Gershman, Horvitz, and Tenenbaum (2015)) hence needs to rely on approximations, shortcuts and heuristics in the presence of limited time and energy – pondering over extended periods of time whether we are absolutely certain that that noise was a dangerous animal was probably not the evolutionary advantageous path. Any approximation has the potential to fail under certain conditions, giving rise to suboptimal behavior. In particular it has recently been suggested that most of experimentally observed behavioural trial-to-trial variability is not the result of uncontrolled intrinsic noise in nervous systems, but can mainly be accounted for by approximate, suboptimal inferences (Beck et al., 2012). The consequences of employing computational shortcuts is well known in the subfield of artificial intelligence that relies on probabilistic computations: approximation algorithms may fail catastrophically for data with particular structure (e.g., Pitkow, Ahmadian, and Miller, 2011). Similarly our brain works well most of the time, but as mentioned above fails to generate reasonable behavior under certain conditions. In our view, deviations from (mathematical) optimality, biases and fallacies of reasoning can provide deep insights into the concrete approximation strategies and heuristics brains of intelligent organisms rely upon, ultimately allowing us to mimic them in artificial agents. A systematic investigation of suboptimal decisions and “fallacies” in human reasoning (see, e.g., Acerbi, Vijayakumar, and Wolpert, 2014) could be of similar importance for theory of brain computation as studies of medical conditions early on for theory of brain organization – we might just learn most about how something

³Generally attributed to John Maynard Keynes.

works when it breaks. In the next section we introduce approximation methods that allow probabilistic computations with distributions of many random variables in systems with restricted computational power.

1.2 Approximate inference

To simplify the discussion, let us first focus on binary random variables (RVs), i.e., events that can either be true or false. This of course does not mean that their chance of occurrence necessarily is all nor none, i.e., 1 or 0. For example the statement “it will rain tomorrow” can turn out to be either be true or false, but your subjective probability that it will rain tomorrow can be anywhere between zero and one:

$$0 \leq p(\text{will rain tomorrow}) \leq 1.$$

To represent a distribution p over all possible states of n binary RVs, we need to store 2^n probabilities, i.e., the memory requirement increases exponentially with the number of RVs. In addition, learning such a distribution would be subject to severe overfitting given the comparatively limited data we have available (Goodfellow, Bengio, and Courville, 2016). In addition, any computation required for exact probabilistic inference is computationally infeasible for a large number of RVs.

One important computation is marginalization, i.e., integrating out a subset of, in this case continuous RVs:

$$p(x) = \int dy p(x, y). \quad (1.1)$$

Imagine representing different physical properties of a three-dimensional object via a distribution, but being only interested in the probability that it has a certain color:

$$p(\text{color}) = \sum_{\{\text{shapes}\}, \{\text{weights}\}, \dots} p(\text{color}, \text{shape}, \text{weight}, \dots).$$

Since we need to sum over all possible values for shape, weight, etc., marginalizing over k binary RVs involves summing over 2^k states, i.e., scales exponentially in the number of RVs. To successfully implement plausible reasoning, intelligent organisms must have evolved sophisticated approximation techniques for computing the relevant quantities and bypassing these computational hurdles. In machine learning and artificial intelligence, the idea to develop shortcuts for probabilistic inference is known as approximate inference (e.g., Bishop, 2006). One particularly successful approach to approximate inference are sampling methods in which one replaces exact computations with stochastic approximation algorithms, so called Monte Carlo methods (Metropolis and Ulam, 1949). The basic idea is that one can learn about a complicated system by simulating it repeatedly, for example using random initial conditions, representing a lack of knowledge about the system. Originally perceived by Stanislaw Ulam in 1946 while playing Solitaire (Eckhardt, 1987) this idea has a long history and found its first heavy and automated use during the Manhattan project to investigate neutron scattering (see Dyson, 2012, for a historic account).

Monte Carlo sampling approaches have a direct link to integration: to compute an infeasible integral, one can rely on the law of large numbers to replace integration with an empirical mean over a large number of samples. Consider for a moment the purely technical issue of calculating the average of some observable $\mathcal{O}(x)$ over a

(possibly multivariate) continuous random variable x with distribution $f(x)$:

$$\langle \mathcal{O}(x) \rangle = \int dx \mathcal{O}(x) f(x). \quad (1.2)$$

Instead of analytically computing such an integral, we could choose a numerical integration scheme with stepsize(s) Δx , i.e., using a grid to approximate the integral using Riemann sums:

$$\langle \mathcal{O}(x) \rangle \approx \Delta x \sum_{i=0}^n \mathcal{O}(x_i) f(x_i). \quad (1.3)$$

This could, for example, be required for a prediction task in which we construct a probabilistic model based on previous observations. Let $\mathbf{X} = (x_0, \dots, x_n)$ represent our previously observed data, and θ the parameters of our probabilistic model that are sufficient statistics for predicting a new value x . Prediction of a new as yet unobserved datum x_{n+1} based on the previously observed data $\mathbf{X}$ would then take the form

$$p(x_n | \mathbf{X}) = \int d\theta p(x_n | \theta) p(\theta | \mathbf{X}).$$

While an approach based on equation (1.3) can work well for a small number of dimensions of x , the number of functional evaluations increases exponentially with the number of dimensions, quickly becoming computationally infeasible. The Monte-Carlo approach instead would be to generate sample points x_i that are distributed according to $f(x)$ and take the empirical mean of the observable over these samples:

$$\langle \mathcal{O}(x) \rangle \approx \frac{1}{N} \sum_{\substack{i=1 \\ x_i \sim f(x)}}^N \mathcal{O}(x_i) =: \hat{\mathcal{O}}(x) \quad (1.4)$$

We use $\sim$ to denote “distributed as” and hats denote approximations obtained via sampling (“estimators”). Sampling methods have the advantage, that the number of function evaluation required to achieve a desired accuracy, i.e., error of the estimator, only depends on the variance of the observable itself, not on the dimensionality of the space (more concrete: the variance of the estimator is proportional to $1/n$ where n is the number of independent samples) (MacKay, 2003). Sampling methods are hence particularly useful in very high dimensional spaces. In addition, many approaches can even be used for densities that are only known modulo a multiplicative constant, i.e., for which we cannot compute the normalization factor.

The challenge in applying Monte Carlo methods is to generate samples x_i that are distributed according to the target distribution $f(x)$. For simple distributions we have full knowledge of $f(x) = f^*(x)/Z$, including the normalization constant $Z = \int dx f^*(x)$ and its cumulative distribution function $F(x) := \int^x dx' f(x')$ with its inverse $F^{-1}(y)$ usually referred to as the quantile function. In this case we can generate uniform random numbers between zero and one ($Y \sim U_{[0,1]}$) and for a realization y_i set $x_i = F^{-1}(y_i)$ which will be a valid sample from f (see, e.g., Robert and Casella, 2005). This method, known as direct sampling, is very powerful, for example to generate normally distributed variables from uniform distributed variables using the Box-Muller algorithm (Robert and Casella, 2005). The cases where

we can directly sample from the target distribution are, however, quite rare and as soon as we are concerned with more complex distributions as they may arise in real-world applications, we need to invoke other methods that do not require knowledge of $F(x)$, $F^{-1}(y)$ or the normalization constant, for example rejection sampling (see below, and Robert and Casella, 2005; Krauth, 2006). These approaches only require us to be able to evaluate f^* , i.e., f up to a multiplicative constant, pointwise.

As a simple illustration consider calculating the mean value, denoted by μ , of an exponential distribution in one dimension:

$$f(x) = \lambda e^{-\lambda x} \Theta(x).$$

For this simple case we can analytically calculate the exact result: $\langle x \rangle = 1/\lambda$. But suppose we want to approximate μ using a Monte Carlo method, i.e., compute an estimate of μ . Direct sampling merely requires us to compute the cumulative distribution function ($F(x) = 1 - e^{-\lambda x}$) and its inverse ($F^{-1}(y) = -1/\lambda \log(1 - y)$). For rejection sampling one introduces an additional distribution from which it is easy to generate samples, the so called proposal distribution g . This should satisfy

$$cg^*(x) \geq f^*(x), \forall x \in \text{supp}(f^*), c \in \mathbb{R},$$

i.e., cg^* should provide an upper bound for f^* over the domain of f where it is not zero. For simplicity we use a uniform distribution as the proposal distribution (figure 1.2). Since the exponential distribution has infinite support we need to artificially introduce a cutoff at $x_{\max}$, i.e., $g(x) = U_{[0, x_{\max}]}$. The maximum of f^* is at $x = 0$ and we accordingly set $c = f^*(0) = 1$. To generate a sample from f , one employs the following procedure. After generating a random sample from g , call it x' , we sample a random value uniformly from $[0, cg^*(x')]$, call it y' . If $y' < f^*(x')$, x' is a valid sample from f , otherwise x' is rejected and the procedure is repeated until a valid sample results.

A comparison of direct sampling and rejection sampling shows that both methods quickly achieve a good estimate of the true value (figure 1.2). Note however the number of accepted samples is much smaller than the number of proposed samples: for this naive choice of proposal distribution a much larger number of values have been proposed during rejection sampling – given a cutoff at $x_{\max} = 200$, the acceptance rate, i.e., the number of proposed samples that lead to a valid sample, is as low as 2%. Rejection sampling is very powerful as it does not require knowledge of Z , however, for high-dimensional spaces, this and similar methods cause severe issues, as the number of rejections increases dramatically (Krauth, 2006). Intuitively this is due to the fact that the volume enclosed by cg^* becomes much larger than the volume enclosed by f^* , and the probability that a random point from cg^* is within f^* becomes extremely small. To sample from distributions in high-dimensional spaces, Markov-chain-Monte-Carlo methods are much better suited.

1.3 Markov-chain-Monte-Carlo methods

Markov-chain-Monte-Carlo methods are a broad class of sampling algorithms including Metropolis-Hasting and Gibbs sampling (MacKay, 2003; Robert and Casella, 2005; Krauth, 2006). The basic idea is to propose new samples based on the current sample, or “state” in this context, and accepting them with a certain probability. However, in contrast to rejection sampling, also proposals for which the new state

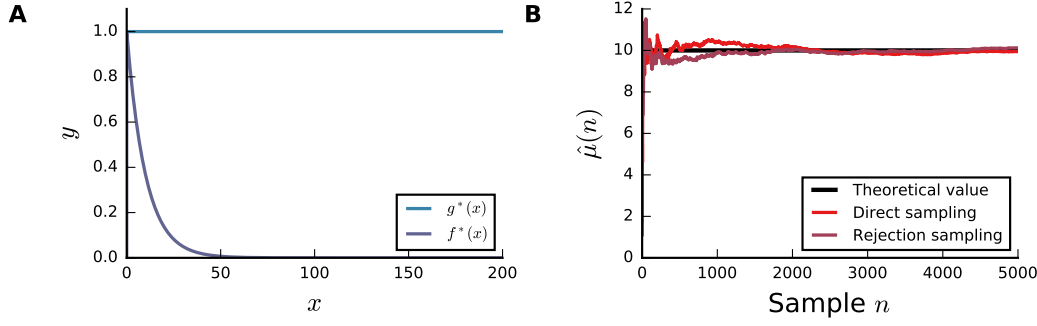


FIGURE 1.2: Direct sampling and rejection sampling approximate the theoretical mean of an exponential distribution. **A:** Non-normalized target distribution f^* and non-normalized proposal distribution g^* . **B:** Running average for the estimator $\hat{\mu}$ over 5000 sampling steps.

is not accepted generate valid samples from the distribution, making them very inefficient even in high-dimensional spaces.

All algorithms of this class possess the property that the conditional probability of the next state only depends on the current state, not on the history of all states:

$$p(x_{k+1}|x_k x_{k-1} \dots) = p(x_{k+1}|x_k). \quad (1.5)$$

This memorylessness of the algorithm is generally known as the Markov property in the theory of stochastic processes (Gardiner, 1985) and the update procedure generates a Markov chain. The conditional probability in equation (1.5) $T(x_{k+1}, x_k) := p(x_{k+1}|x_k)$ is referred to as the transition kernel of the Markov chain. In contrast to the previously introduced methods, samples generated via this approach are correlated. This causes severe problems in the error analysis of the estimator, since the law of large numbers does not hold trivially any more (see, e.g., Krauth, 2006). In the framework of Markov processes, it can be shown that under certain assumptions⁴ each Markov chain possesses a unique stationary distribution of states to which it converges for a large number of samples (Gardiner, 1985). In other words, after a certain number of transitions from the initial state (sometimes referred to as the “burn-in time”), samples produced by the Markov chain are distributed according to some unique distribution h . For low-dimensional Markov chains, this burn-in time can be estimated from properties of the transition kernel, but for high-dimensional problems, the burn-in time is generally unknown (Krauth, 2006). Therefore, to improve the estimate of the distribution, one typically ignores the first few samples of the Markov chain as they heavily depend on the initial conditions.

The most general example of MCMC algorithms is the Metropolis-Hastings algorithm (Hastings, 1970). In this algorithm one defines a transition kernel $T(x', x)$ that determines the transition probability from state x to x' to construct a Markov chain such that its stationary distribution h is identical to some distribution f from which we want to generate samples, i.e., $f \equiv h$. Starting from an arbitrary initial state x the algorithm proceeds by repeatedly proposing a new state x' given the current state x via $T(x', x)$ and accepting x' with probability

$$a(x', x) = \min \left(1, \frac{f^*(x')T(x, x')}{f^*(x)T(x', x)} \right). \quad (1.6)$$

⁴(i) existence of a stationary distribution, with detailed balance as a sufficient criterion (ii) ergodicity of the chain, i.e., every state can be reached from any other and no state is periodic

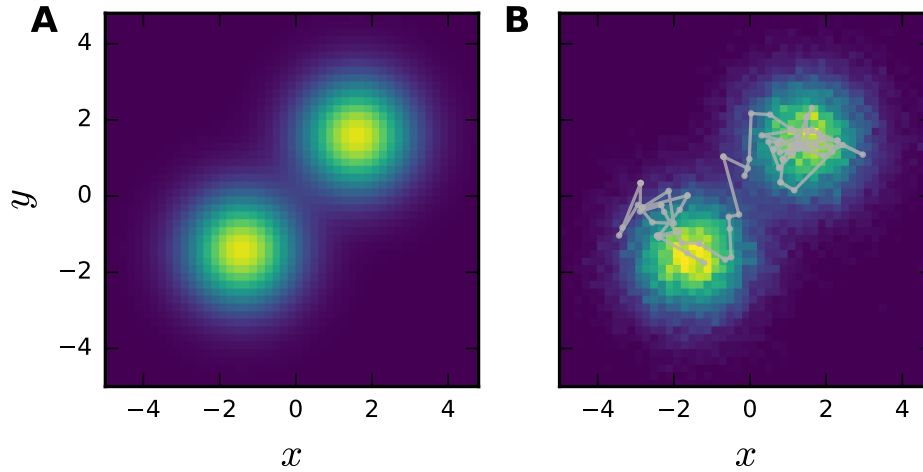


FIGURE 1.3: Metropolis sampling approximates the true probability distribution for a mixture of Gaussians. **A:** Theoretical density f . **B:** Empirical density obtained via Metropolis sampling with Gaussian transition kernel. Empirical distribution obtained from 10^5 samples. Light-gray dots and lines indicate 90 subsequent samples from the end of a trial.

Note that also in this method we only need to know the functional form of the distribution f up to a multiplicative constant, making it applicable in a wide variety of circumstances. If we choose T to be symmetric, i.e., $T(x', x) = T(x, x')$, $\forall x, x'$, the formula for the acceptance probability simplifies to

$$a(x', x) = \min \left(1, \frac{f^*(x')}{f^*(x)} \right), \quad (1.7)$$

corresponding to the original algorithm first introduced by Metropolis et al. (1953). From equation (1.7) one can intuitively see how the algorithm is more likely to transition from low-probability to high-probability regions than vice versa and hence should produce more samples x for which $f(x)$ is large. If one rejects a transition ($x \rightarrow x'$), the current state x is not changed but again used as a valid sample from f , one source of correlations between subsequent samples. The problem of correlated samples is often addressed by collecting n samples, but considering only every k th sample, so called “thinning” of the Markov chain. Despite issues of indetermined burn-in time and correlations between subsequent samples, MCMC methods are at the heart of many modern applications that involve multi-dimensional integrals, for example in statistical physics (e.g., Krauth, 2006) or Bayesian data analysis (e.g., Gelman et al., 2014).

The correlation between samples arising in approximate inference algorithms employing MCMC methods gives rise to systematic biases of the estimator. Consider for example a bimodal distribution from which we would like to generate samples. Depending on the initial conditions of the sampler and an appropriate step size, the algorithm will initially explore only one of the modes, i.e., generate samples that are similar to the initial guess. As time passes, the sampler will also explore the second mode, and will start to alternate between the two modes, on a timescale typically larger than the time between two subsequent samples (figure 1.3). These dynamics generate typical signatures that can be related to animal behavior. For example when viewing visual stimuli with different interpretations, so called multi-stable stimuli, perceptions of subjects and neural activity changes between possible interpretations,

with subjects maintaining the current percept for extended but stochastic periods of time (Blake and Logothetis, 2002; Sundareswara and Schrater, 2008; Gershman, Vul, and Tenenbaum, 2012). Furthermore, researchers have found that the average of two guesses about real-world knowledge from the same person has a smaller error than an individual guess – in particular when the two guesses were long apart, since the errors in estimates were less correlated (Vul and Pashler, 2008; Denison et al., 2013). Other studies have suggested that the anchoring bias, i.e., the observation that guesses of individuals about unknown quantities tend to stay close to initially provided, external cues, can be explained by resource-constrained approximate inference relying on samples obtained during the burn-in time of a Markov chain (Lieder, Griffiths, and Goodman, 2012).

If the number of random variables and hence the dimensionality of the state space is high and new samples are proposed along arbitrary directions of the state space, it can be difficult to find an appropriate transition kernel that leads to good mixing, i.e., short burn-in time and small autocorrelations of subsequent samples. An alternative approach that only updates one direction at a time while keeping the others constant is Gibbs sampling. This method is applicable when the conditional distributions of single variables are known and simple to sample from, leading to a smaller range of applications, but higher efficiency in the cases where it can be applied (Robert and Casella, 2005). Starting from some initial state vector $\mathbf{x}$, the algorithm repeatedly generates new values sequentially along all directions of $\mathbf{x}$, updating one component at a time according to $f(x_i | \mathbf{x}_{\setminus i})$. Here $\mathbf{x}_{\setminus i}$ denotes the vector $\mathbf{x}$ without the i th entry. This algorithm is of particular interest in computational neuroscience, since for particular probability distributions models have been proposed that are amenable for implementation in neural substrates (Hinton, Sejnowski, and Ackley, 1984a; Buesing et al., 2011). In these models it is possible to interpret the updates of the Markov chain as stochastic dynamics of a network of interconnected simple units, and to approximately map the algorithm to dynamics of biophysically plausible neural networks (Petrovici et al., 2016) thus providing a basis for approximate probabilistic computations in nervous systems. We will discuss this further in section 2.5.

These approximations of probabilistic inference described above are a promising path from their high-level description to their low-level implementation. Through the possibility of relating signatures of neurally plausible approximate inference methods to results from cognitive science and neuroscience, the Bayesian brain hypothesis is not only a macroscopic theory of how humans behave but is concerned with all levels of description that Marr and Poggio (1976) defined as a prerequisite to understand neural information processing:

- **Computational level: probabilistic inference**

The probability calculus as an extended form of logic, obtained from consistency with common sense and internal consistency requirements, is the appropriate tool to derive conclusions from uncertain statements and hence a normative model of brain function.

- **Algorithmic level: resource-constrained approximate inference**

Since exact inference is infeasible even for moderate number of random variables, systems with finite computational power employ approximation methods, for example Markov chain Monte Carlo sampling.

- **Physical level: recurrent networks of nerve cells**

The main cognitive processes are reasonably believed to arise from interaction of large numbers of nerve cells and their interconnections. Via stochastic network dynamics they provide a basis, for example for sample-based approaches to approximate inference.

Understanding the concrete implementation details, the “nuts and bolts” of brain function, is not only interesting for basic science, but also relevant for an application driven, machine learning point of view. The brain is currently the best-functioning real-world approximate-inference machine and the only physical system that we know capable of what we would call general intelligence: should we ignore its details even if our focus would be on designing smart(er) machines? Imagine a machine bestowed on us by an advanced alien civilization that is capable of everything that we would refer to as intelligent behavior. Despite its most likely daunting complexity, we should not deny that studying this apparatus in all its mechanistic detail is a very promising way to construct such a machine ourselves. To be able to discuss the low-level implementation of brain function we now explore biophysical aspects of nervous systems.

Chapter 2

Computational neuroscience

The language of the brain is based on neurons. To understand the brain you must understand neurons and especially how vast numbers of them act together in parallel.

Francis Crick, The Astonishing Hypothesis, 1994

The field of neuroscience focuses on the study of structure, dynamics and function of networks of biological cells that form what we usually refer to as the central (brain and spinal cord) and peripheral nervous system. Not only are these systems responsible for keeping us alive, perceiving and navigating in the world, but are believed to be the very essence of what makes us human: all high-level cognition and consciousness (see, e.g., Crick and Koch, 1990; Crick, 1995). The motivation to study this complex system hence comes from many different field with questions such as:

- Biology: How is the brain organized? How did it evolve? How does it change during the lifetime of organisms? How does it differ between species?
- Medicine: What is the effect of damage and what are means of rehabilitation? What are the origins and effects of neurodegenerative diseases (e.g., Alzheimer's, Parkinson) and how can they be treated effectively?
- Technology: How does the brain compute/learn? What are its information-processing principles? How can we built smart, energy-efficient and fault-tolerant machines following similar principles?
- Philosophy: How can immaterial things like thoughts and feelings have an influence on such physical things as a human body? What is consciousness and where can we find it?

This variety illustrates the necessarily strong interdisciplinary character of modern neuroscience.

The dynamics of biological neural networks span a wide range of time and length scales from the (sub)millisecond and (sub)micrometer involved in processes at individual synapses, to months and years over which large-scales structures develop. At present all of these seem to be relevant. It is hence generally recognized that we will not be able to understand brain function by focusing on a single scale, but need to consider descriptions at a variety of interrelated scales (Marr and Poggio, 1976). Accordingly experimental techniques to investigate nervous systems have been developed for almost all temporal and spatial scales (figure 2.1). These range from ultra-high resolution imaging techniques of dissected tissue using electron microscopy, via electrophysiological techniques like patch clamp recordings or optogenetic methods like calcium imaging in living cells, all the way up to fMRI, PET and

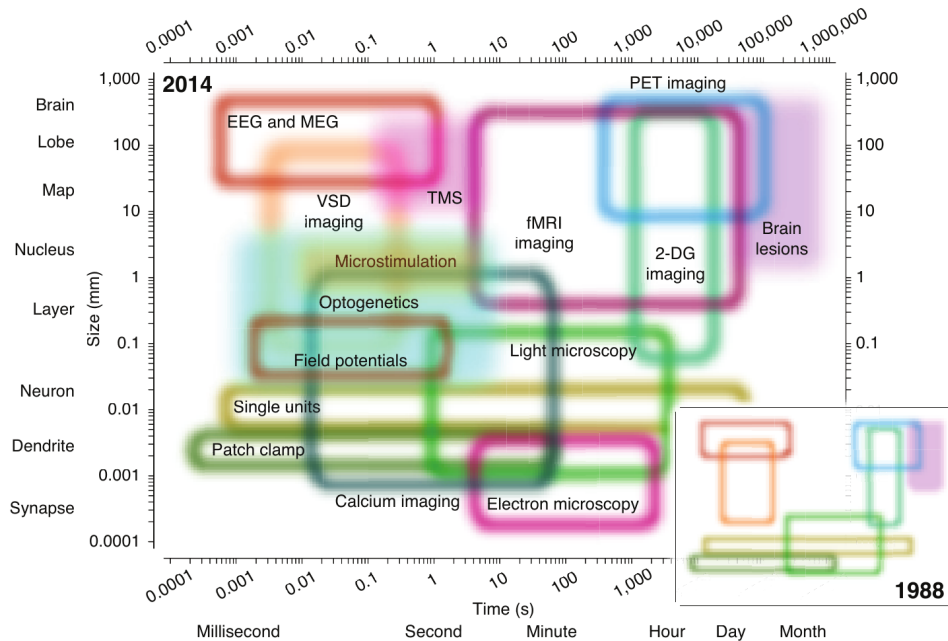


FIGURE 2.1: The study of nervous systems spans a wide range of different time and length scales covered by various experimental techniques. Reprinted by permission from Macmillan Publishers Ltd: Nature Neuroscience (Sejnowski, Churchland, and Movshon, 2014), copyright (2014).

other imaging techniques on the systems level (Kandel, Schwartz, and Jessel, 2000; Buzsáki et al., 2015). To deal with the complexity of data acquisition and handling, modern neuroscience involves huge collaborative efforts, supported by large-scale initiatives like the Human Brain Project (HBP) and the Brain Research through Advancing Innovative Neurotechnologies (BRAIN) initiative (Kandel et al., 2013). The technological relevance of deciphering the working principles of neural systems is demonstrated, for example, by large technology companies teaming up with US institutions to develop biologically inspired computing hardware under the SyNAPSE program (DARPA, 2008).

The central aim of neuroscience, as developed in scientific investigation over the past 200 years, is to relate the behavior of an organism to the structure and dynamics of its nervous system. Modern neuroscience rests on the hypothesis that we can understand the behavior of intelligent animals and in particular the human mind, including abstract concepts like desires, feelings and consciousness, purely with reference to the biophysical basis of individual molecules and cells without reference to additional (metaphysical) mechanisms or “special sauce” as Hawkins and Blakeslee (2004) put it.

Here we will constrain our discussion to the mammalian brain and specifically to the cerebral cortex, the youngest part of the mammalian nervous system. Cortex is the surface of the brain, sometimes referred to as the “gray matter” due to its characteristic color after conservation, in contrast to the “white matter” below the cortex mainly housing long-range fibers connecting populations of nerve cells. Cortex has a folded structure, allowing to host more cells in the available space between the skull bones. The reasons for focusing on cortex are manifold. Nerve cells, in the following also referred to as neurons, in cortex are arranged in so called cortical columns. These are highly coupled populations of neurons within cylindrical volumes with stereotypical connectivity. This structure is observed repeatedly all across cortex

Motor and Sensory Regions of the Cerebral Cortex

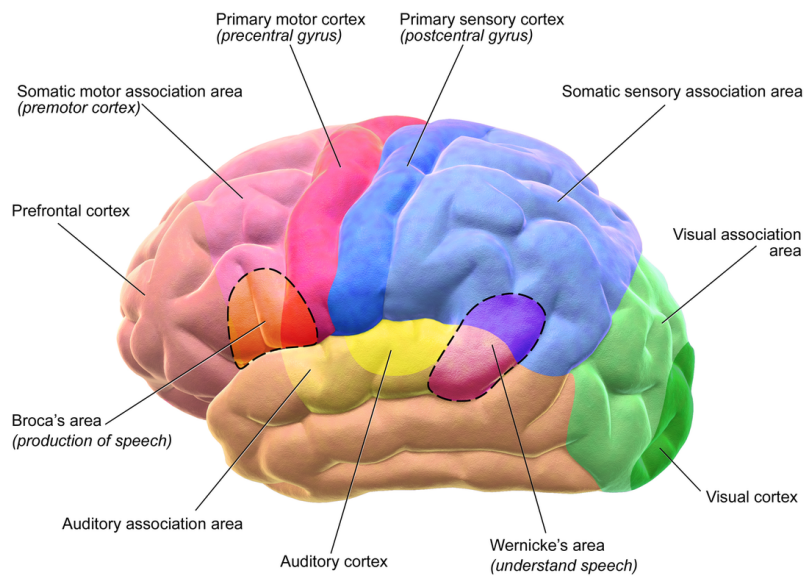


FIGURE 2.2: An inflated surface map of the cerebral cortex, showing its subdivision into functional regions (Blausen.com, 2014).

giving rise to the notion of individual columns as “basic functional units” (Mountcastle, 1978; Dayan and Abbott, 2001). In addition, neurons from different sensory cortices can take on new function when provided with sensory input different from what they originally received, e.g., areas originally responsible for visual perception becoming responsive to somatosensory input in blind subjects (see, e.g., Cohen et al., 1997; Bavelier and Neville, 2002). Together these observations inspire the view of cortex as a universal computing substrate with repeated stereotypical computational structures from which we can learn about the intricacies of brain computation. Cortex is also reasonably believed to be the origin of higher cognitive functions, e.g., language, planning, and consciousness (Kandel, Schwartz, and Jessel, 2000) making it an attractive system to study. Pragmatically, we can view cortex as toy system for development of computational and mathematical tools while assuming that most methods and models will be transferable to other neural systems.

The mammalian cortex can be divided into four anatomically distinct lobes, each mainly responsible for a specific set of functions: frontal (planning and motor control), parietal (somatic sensation), occipital (vision), temporal lobe (hearing). Early clinical studies have shown that damage to particular well defined regions can result in loss of certain cognitive capabilities in humans. For example one can delineate areas, that broadly speaking, if damaged, can lead to loss of the ability to speak properly (Broca’s area) (Broca, 1865) or understand language (Wernicke’s area) (Wernicke, 1906) (figure 2.2). Following these initial observations researchers created fine subdivisions of cortex based on functional and anatomical features, e.g., Brodmann (1909). Nowadays it is generally accepted that (higher) cognitive functions arise from interaction between different cortical areas and sub-cortical brain regions (Kandel, Schwartz, and Jessel, 2000; Bressler and Menon, 2010).

Based on the subdivision of cortex into separate areas, it is possible to formulate a

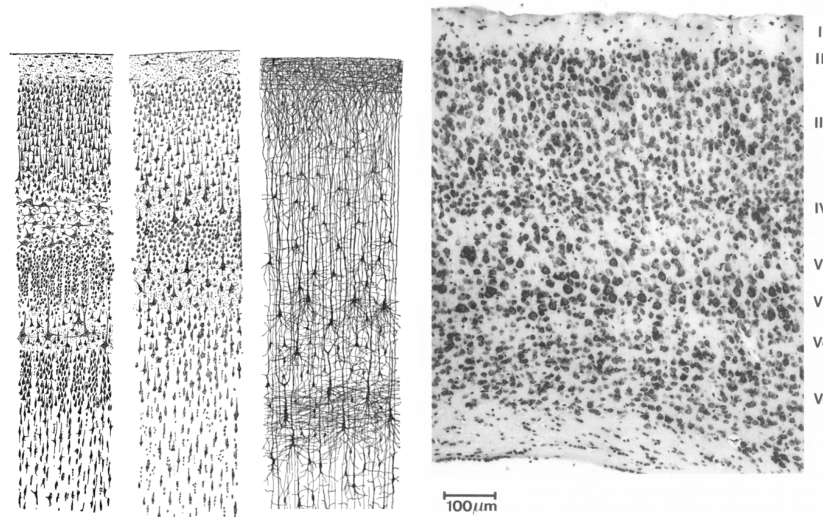


FIGURE 2.3: Layered structure of cortical slices with layer-specific neuron types and neuron density. Left: neurons in the auditory cortex, drawn by Cajal (1899). Right: Nissl stain of mouse visual cortex. Reprinted from Braitenberg and Schüz (1991) with permission of Springer.

hierarchy according to how close an area is to input (sensory) or output (motor) areas (Felleman and Van Essen, 1991). This hierarchy gives rise to the notion of bottom-up from lower to higher areas and top-down from higher to lower areas (Kandel, Schwartz, and Jessel, 1991). Inference in vision for example could be achieved by combining feedforward (evidence) with top-down input (prior expectation), generally known as the predictive coding hypothesis (e.g., Mumford, 1992; Rao and Ballard, 1999). While researchers initially focused on the differences between areas, it was soon realized that the similarities are more striking than the differences (Mountcastle, 1978): Cortex is remarkably similar across areas, individuals and even species according to measures like thickness, vertical organization, cell densities and properties of local connectivity (Abeles, 1991; Braitenberg and Schüz, 1991), further fueling the hope for a general working principle of cortex.

We now zoom in from the macroscopic level to a small piece of cortex, about 1 mm^3 in volume. One of the earliest detailed depiction of the structure of such a small piece was created by Ramon y Cajal, who used so called silver staining to stain neurons in a small slice of neural tissue and captured the resulting structures in detailed drawings (figure 2.3). They reveal a layered structured of cortical tissue: at different depths, one finds different densities and types of nerve cells. This was first evidence that our brains are highly complex, structured networks of discrete, interconnected biological cells with complex spatial arrangements. Modern evidence confirms these findings and in addition reveals that projections from/to remote areas predominantly target/originate from specific layers (Kandel, Schwartz, and Jessel, 1991; Markov et al., 2014b).

The entire human brain consists of about $80 \cdot 10^9$ neurons, with about 20% of them located in the cerebral cortex (Azevedo et al., 2009; Herculano-Houzel, 2012) and about 10^5 of them found under a 1 mm^2 of cortical surface. Even more impressive than these cell counts is the network of connections between neurons: staining a certain part of the connections in a small slice of cortex demonstrates that there is little else than wiring (figure 2.4). It is very difficult to estimate the number of connections between neurons, but based on the finding that each neuron in cortex

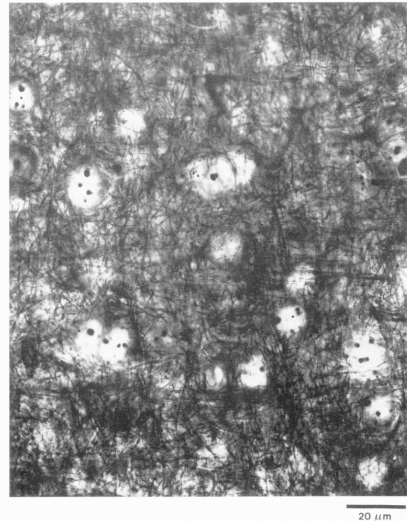


FIGURE 2.4: Dense connections between neurons form an intricate network. Silver stain of all axons in a piece of layer V of mouse cortex. Reprinted from Braitenberg and Schüz (1991) with permission of Springer.

receives up to 10^4 connections from neighboring as well as remote neurons (Abeles, 1991), the estimated total number of connections in human brains is 10^{14} (Roth and Dicke, 2005). Making sense of this complexity is an enormous undertaking.

To support experimental investigation by compressing knowledge and abstracting away complexity, researchers introduced mathematical and computer-aided modeling of the nervous system and automated analysis of experimental data, creating the field of computational neuroscience (see Schwartz, 1990). Mathematical and computational models have the significant advantage over experiments that they provide full control over all parameters of the model, something that is impossible to realize in experimental procedures. In addition, simulations do not suffer from the issue of undersampling: Even with modern techniques only a minuscule part of the whole network can be observed, with the possible number of simultaneously recorded neurons currently in the order of 100 – 1000 and doubling approximately every seven years (Stevenson and Kording, 2011). As in all natural sciences, mathematical and/or computational models are required because the real world is too complex to understand by observation alone – abstraction and simplification allows us to extract and understand fundamental principles.

The immensely complex connectivity of cortical networks forces us to introduce many simplifying assumptions to model and analyze neuronal systems. We need to decide which parts of biological observations we will take into account and which we will neglect. While making the necessary abstractions, the main goal is to create models compatible with available evidence that allow predictions for future experiments, thus creating loops of interaction between theory and experiment.

2.1 Defining the microscopic level

The purpose of physical sciences is to find an appropriate mathematical description of reality that allows us to formally describe observed phenomena and predict experimental results under previously unobserved conditions (a form of “scientific inference” (Jeffreys, 1973; Jaynes, 2003a)), making these proposed descriptions falsifiable (Popper, 1935). To develop a coherent theory of the brain, we need to define

what aspects of nervous systems we will describe and which parts we will ignore for the present investigations. Here, we need to decide on the microscopic level to model the nervous system, facing a tradeoff between mathematical tractability and predictive power. Should we choose a description in terms of areas, populations, cells, molecules or subatomic particles? The appropriate level heavily depends on the phenomena that we want to capture in our theory. As mentioned before we are mainly concerned with understanding how the interactions of a large number of individual neurons organize in complex networks to give rise to macroscopic dynamics and function, i.e., behavior of an organism. This naturally leads us to a level of description in terms of individual nerve cells and their connections, each of which we will model using very simple dynamical descriptions as discussed in section 2.2. In analogy to statistical physics, where molecular dynamics are neglected in favor of a simplified description sufficient to describe the macroscopic properties of a large collection of individual molecules, we neglect many microscopic details of biophysical processes in individual cells.

This underlying hypothesis is that the relevant dynamics in neuronal networks mainly arise due to the complex connectivity, and not due to complex single cell dynamics, a view usually referred to as “connectionism”. In the field of machine learning this view has led to several breakthroughs in the last decade through “deep learning” techniques (LeCun, Bengio, and Hinton, 2015; Goodfellow, Bengio, and Courville, 2016) which were inspired by observations in biological neuronal networks. By introducing such models, we adopt a mechanistic/reductionist world view: We aim to explain the complex whole in terms of smaller, well-defined and simpler parts. In particular we understand all observed macroscopic dynamics of organisms as emerging properties of the physical interaction of large numbers of biological cells. This abstraction only considers phenomenological models of neuronal dynamics and it is possible that additional details of biological processes need to be included in the theory. We now describe the elementary building blocks of neuronal networks and mathematical models that we will use to describe them.

2.2 Single neurons

In the following two sections, we introduce some biophysical properties of nerve cells, in particular their communication through short electrical pulses, and afterwards introduce mathematical and computational models that we will use to investigate networks of such cells.

2.2.1 Biophysics

In simple terms, neurons consist of three parts: dendrites (the “antennas”), soma (the “integrator”) and axon (the “transmitter”) (figure 2.5) (Kandel, Schwartz, and Jessel, 1991). The dendrites of neurons form complex tree-like structures with an entire length of about 4 mm per neuron in mouse cortex. The axon can be idealized as a long cable that extends on average for about 40 mm connecting to the dendritic trees of other neurons. Both are connected to the soma, which in a typical cell is on the order of $10 - 100 \mu\text{m}$ wide (Dayan and Abbott, 2001). Neurons come in a variety of shapes, arrangements of dendrites and axon, and molecular composition, leading to between a hundred to a thousand different types that can be distinguished in cortical tissue (Kandel, Schwartz, and Jessel, 2000; Markram et al., 2015).

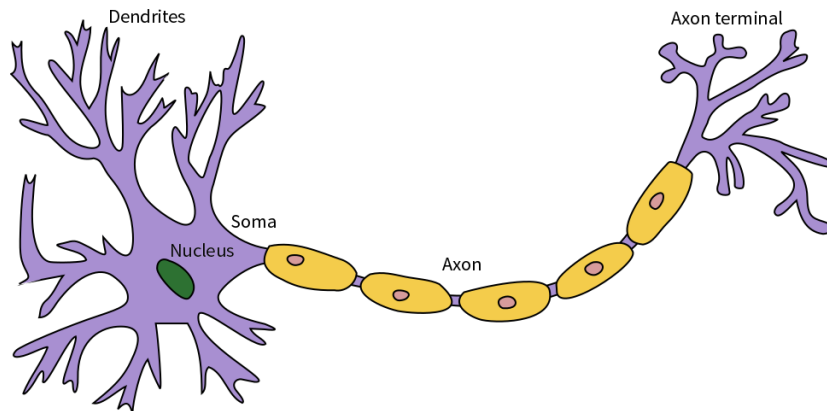


FIGURE 2.5: Simplified sketch of the most important components of a neuron. See text for details. Modified from Wikipedia.org.

Neurons have a cell membrane: a lipid layer a few nm thick that separates the intracellular medium from the extracellular space. The main feature of neurons that distinguishes them from other biological cells is the large voltage difference between inside and outside. This difference arises from a separation of ions across the membrane that is maintained by a combination of active and passive ion channels embedded in the lipid layer (for details see Kandel, Schwartz, and Jessel, 1991). By perturbing this potential, the cell is able to actively propagate electrical signals and thereby communicate with other cells (see below). The most important ions involved in these processes are Sodium (Na^+), Potassium (K), Calcium (Ca^{2+}) and Chloride (Cl^-). Active ion pumps create a higher concentration of K inside the cell and higher concentration of Na^+ , Ca^{2+} in the extracellular medium (figure 2.6). By inserting an electrode into a neuron, for example in an isolated slice of living tissue, one can record the potential difference between the inside and outside the cell. This is referred to as the membrane potential which at rest, i.e., in the absence of any external inputs, is around -70 mV with respect to the extracellular medium. Due to the large spatial extent of a neuron, the membrane potential measured in different parts of the same cell can be very different (Williams and Mitchell, 2008). However ionic currents flowing between different parts of the cell tend to equilibrate these differences. Movement of ions across the membrane leads to current flow in/out of the cell, which depending on their direction and charge make the membrane potential more negative (hyperpolarized) or less negative (depolarized). Due to voltage sensitivity and specific temporal characteristics of the ion channels, a significant depolarization can lead to rapid excursions of the membrane potential after reaching a certain threshold potential. These excursions, called action potentials or spikes, have stereotypical shape with a magnitude of about 100 mV , last for about 1 ms (figure 2.6) and are initiated at the point of the cell where the axon is connected to the cell body. In contrast to subthreshold fluctuations that are attenuated over a few mm, these action potentials actively propagate down the axon until they reach the axon terminal, the point where neurons connect to other cells via chemical synapses. Note that in biological tissue, these action potentials do not only propagate down the axon, but can also influence the potential in the dendritic tree (backpropagating action potential), playing an important role in the process of learning. Action potentials are the main form of communication in the nervous system. Sensory information, for example from our eyes or ears, uses the same means of communicating sensory stimulation to the central nervous system. For the brain, there is no fundamental difference

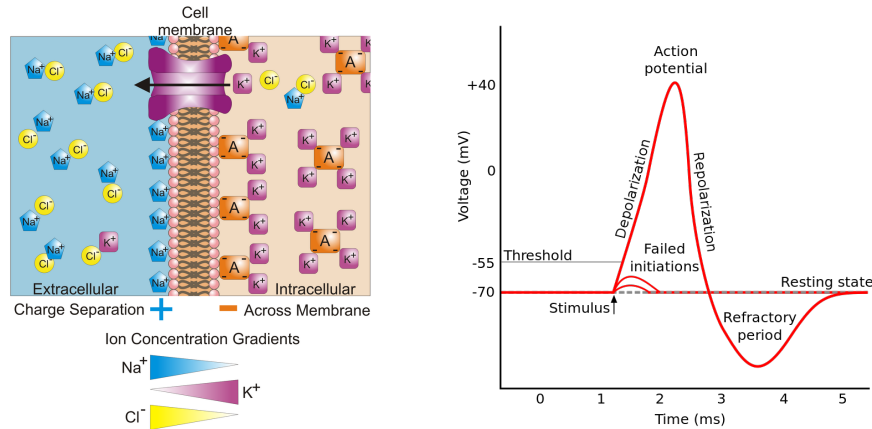


FIGURE 2.6: Left: neuronal cell membrane with embedded ion channel. Right: membrane potential as a function of time for different current stimuli. Modified from Wikipedia.org.

between visual or auditory information except the target areas for projections from the various senses.¹ When an action potential reaches the axon terminal, the rapid change in potential leads to influx of Ca^{2+} into the cell and triggers the fusion of vesicles containing neuron-specific molecules, so called neurotransmitters, with the cell membrane. Neurotransmitters are released into the space (“synaptic cleft”) between the source (presynaptic) and target (postsynaptic) neurons (figure 2.7) and diffuse through the extracellular medium. Upon reaching the membrane of the postsynaptic cell they can interact with specific receptors in the membrane, causing currents to flow in and out of the postsynaptic cell (postsynaptic currents) that lead to a change of the membrane potential in the postsynaptic neuron (postsynaptic potential). All of these processes take a finite amount of time summing up to delays between 0.1 and 10 ms from eliciting an action potential at the soma in a presynaptic neuron to a change in the membrane potential in the dendrite of a postsynaptic cell. These synaptic delays need to be taken into account in neural network models since they are important for network dynamics and function, for example for the generation of oscillations (Brunel, 2000; Helias, Tetzlaff, and Diesmann, 2013) or for enabling sound localization via coincidence detection (Jeffress, 1948). While different neuron types can release different types of neurotransmitters that have specific influences on postsynaptic cells, a single neuron typically releases a specific set of neurotransmitter at all its synapses. In the mammalian brain, neurotransmitters either have excitatory (depolarizing; typically glutamate) or inhibitory (hypolarizing; gamma-aminobutyric acid, GABA) effects on the postsynaptic membrane potential (Kandel, Schwartz, and Jessel, 1991). This allows us to make a general classification of neurons into excitatory and inhibitory cells, generally known as Dale’s principle (Eccles, P, and Koketsu, 1954). While excitatory neurons form short- and long-range connections, connections from inhibitory neurons are typically restricted to a small, local region. A single excitatory postsynaptic-potential increases the membrane potential of the postsynaptic cell by about 0.1 – 1 mV. With a potential difference on the order of 10 mV between the resting and threshold potential, a neuron hence needs to receive between 10 – 100 excitatory inputs in close temporal proximity to elicit a spike itself, since the influence of a single postsynaptic potential only lasts between 10 – 100 ms, depending on the type of neurotransmitter involved (Kandel, Schwartz,

¹See for example Hawkins and Blakeslee (2004) for a discussion of the remarkable implications of this fact and Thomson, Carra, and Nicolelis (2013) for experimental results.

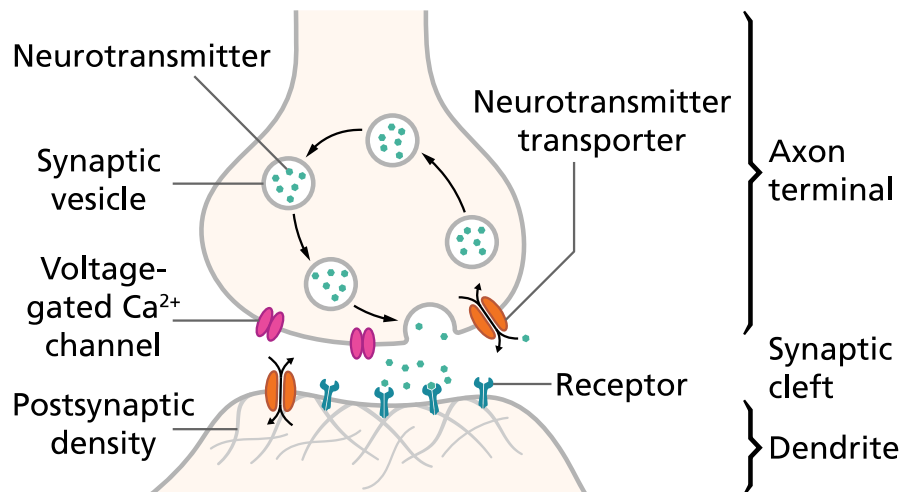


FIGURE 2.7: Simplified sketch of a chemical synapse (Spletstoeser, 2017).

and Jessel, 1991). Excitatory neurons outnumber inhibitory neurons by a factor of 4 : 1 (Braitenberg and Schüz, 1991), however, inhibitory neurons are generally more active and have denser projections, thereby stabilizing the network (Petersen and Crochet, 2013).

In addition to chemical synapses, other types of connections between neurons exist, where the intracellular media of two neurons are bi-directionally coupled via channels in their membranes, so called gap junctions. Although gap junctions are significantly outnumbered by chemical synapses, they can significantly influence global network dynamics (Dere and Zlomuzica, 2011).

Compared to the direct coupling via gap junctions, the detour via chemical junctions might seem cumbersome. However, chemical synapses allow for a plethora of effects at the coupling site that influence synapse, neuron and network dynamics. This includes changes in the coupling strength between two neurons, so called synaptic plasticity, that play an important role for learning, for example short-term, long-term and neuromodulated plasticity. The earliest principle for learning in the nervous system was proposed by Hebb (1967) as a basis for associations: cells that are connected and active in close temporal proximity should increase their coupling strength, since their coactivation may reflect correlations in the stimuli they are sensitive to. In chapter 9 we will encounter a network model that relies on a slight modification of this rule to model how networks can learn tasks from reward and punishment. It was subsequently observed (Bi and Poo, 2001) that this rule needs to be extended when taking the timing of spikes into account, since neurons tend to increase the strength of their connection when the postsynaptic neuron is active after the presynaptic neuron and decrease it if the order is reversed, giving rise to so called spike-timing-dependent plasticity (STDP) that provides a basis for causal associations. While early models only considered the timing of pre- and postsynaptic spike to be relevant for the actual change of connection strength (Morrison, Aertsen, and Diesmann, 2007b), subsequent models included additional components that influence plasticity, such as neuromodulators or postsynaptic voltages (Clopath et al., 2010; Frémaux and Gerstner, 2015). In addition to these processes producing long term changes in the network structure, changes on a shorter timescales occur, for example, due to depletion of neurotransmitters in the axon terminal (short-term plasticity, e.g., Rosenbaum, Rubin, and Doiron, 2013). While short- and long-term plasticity involves modulation of the connection strength of existing synapses, structural

plasticity leads to the creation of new and removal of old synapses in neuronal networks (Holtmaat and Svoboda, 2009), a process that is not only important during development but is also ongoing in adult brains (May, 2011).

All of these processes involve an inconceivable number of molecular biophysical processes, but we will abstract most of these away and work with phenomenological neuron and synapse models that capture particular effects believed to be important for macroscopic dynamics, with the hope that the abstraction will allow for an easier understanding. These effects include the integration of synaptic input, the generation of action potentials and varying synaptic coupling strengths through plasticity rules. We will also ignore the role of other cells that are found in the central and peripheral nervous system (“glia”), although they are believed to play an important role for the overall function of the system, for example by forming myelin sheets around axons that increases the speed of axonal transmission. The total number of glia cells is comparable to the number of neurons (Azevedo et al., 2009).

Although biological neurons show intricate spatial structures (see, e.g., figure 2.3), we will describe neurons as single points in space by point-neuron models, i.e., we will not describe the ion flow in the intracellular medium. This reduces the description of the membrane potential of a cell to a one dimensional problem. However, recently few-compartment models have gained attention due to the discovery of additional biophysical properties that can influence synaptic plasticity, inspiring new models of synaptic plasticity (Artola, Brocher, and Singer, 1990; Clopath et al., 2010; Major, Larkum, and Schiller, 2013; Urbanczik and Senn, 2014). In the next section we turn to common mathematical abstractions of neurons and synapses.

2.2.2 A mathematical description of neurons and synapses

To analyze, model and understand the behavior of (networks of) individual neurons we need to find a formal description that strikes an appropriate balance between mathematical and computational tractability and biophysical realism. The first mathematical model describing the membrane-potential and ion-channel dynamics of an isolated neuron in response to current stimulation was developed by Hodgkin and Huxley (1952). In this model, the membrane potential is determined by the interaction of two different ion channels regulating the conductances for sodium and potassium. The model abstracts a small patch of membrane as an equivalent electrical circuit that mimics the behavior of the most important biophysical components: ion channels as variable conductances, a constant leak across the membrane as a fixed conductance and the insulating membrane as a capacitor (figure 2.8). Here C_m describes the capacity of the cell membrane, E_n is the reversal potential of a certain ion channel with conductance g_n , E_L is the a potential that the neuron would converge to due to the active ion pumps and leak currents if sodium and potassium channels were closed leading to the conductance g_L . I_p finally lumps together any additional currents, for example due to synaptic inputs from other neurons or stimulation via an electrode. Using Kirchhoff’s law of charge conservation, we can write down the dynamics of the potential across the capacitor for an arbitrary number of ion channels with reversal potentials E_n :

$$C_m \dot{V}(t) = -g_L(V(t) - E_L) - \sum_n g_n(t, V) (V(t) - E_n) + I_p(t). \quad (2.1)$$

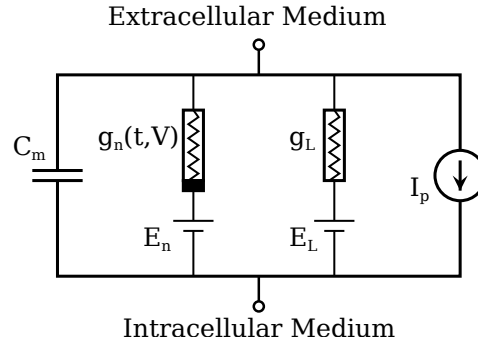


FIGURE 2.8: Equivalent circuit for a model neuron with (from left to right) a capacitance modeling the insulating cell membrane, a voltage-sensitive resistor modeling voltage sensitive ion channels, a constant resistor modeling constant leak current across the membrane and a current source arising from synaptic input or stimulation via an electrode. From Wikipedia.org.

Dots to denote temporal derivatives ($\dot{f}(t) := \partial_t f(t)$). These simple dynamics, supplemented by additional differential equations for the dynamics of the conductances, capture the biophysical behavior, in particular action-potential generation, of biological neurons very well (Hodgkin and Huxley, 1952).

We make a further simplifying assumption, namely that the conductance of the ion channels is not voltage dependent. This removes the dynamics required for action-potential generation, and we hence need to introduce an alternative mechanism to compensate for this: whenever the voltage reaches a certain threshold, we set the potential to the reset potential and send a spike to all neurons that this neuron projects to. In addition, after the neuron has generated a spike, we clamp the membrane potential to the reset value for a short time τ_{ref} , mimicking the observed refractory period during which a biological neuron does not spike even in response to strong inputs (Dayan and Abbott, 2001). It should be noted that while this seems like a strong simplification, these so called threshold models can actually reproduce the spiking behavior of real neurons fairly well (Gerstner and Naud, 2009). We abstract the spike train $\mathfrak{s}(t)$ generated by a threshold model as a sum of Dirac-delta distributions centered at the time of the spikes for mathematical convenience. This reflects that we model communication between neurons purely by the timing t_i of an action potential, not by its specific shape:

$$\mathfrak{s}(t) = \sum_i \delta(t - t_i). \quad (2.2)$$

This very simple model is known as the leaky-integrate-and-fire model, proposed even before the mechanism of action-potential generation was described (Lapicque, 1907; Brunel and Rossum, 2007). The differential equation describing its dynamics can be written as

$$\tau_m \dot{V}(t) = -Rg_L(V(t) - E_L) - R \sum_n g_n(t) (V(t) - E_n) + RI_p(t), \quad (2.3)$$

where we replaced the capacitance $C_m = \tau_m/R$ by introducing the membrane time constant τ_m and membrane resistance R . In this form, interactions between neurons are modeled by presynaptic spikes leading to changes in the conductances $g_n(t)$ of ion channels in the postsynaptic cell, leading to non-linear differential equations for

the membrane potential, so called conductance-based models. One important feature of conductance-based models is the shunting effect of dynamical conductances: the resistance of the cell can be dynamically changed by to the total number of open ion channels.

In vivo, i.e., in a living organism, each cell is under constant bombardment with input from other neurons, causing neurons to enter a high-conductance state (Destexhe, Rudolph, and Pare, 2003). For simplicity, we can hence consider a further simplification by considering our model neurons to be in a high conductance state, i.e., in a in vivo situation. This allows us to assume conductances to be static, and model the interaction between two neurons by the presynaptic neuron directly influencing the current flowing into the postsynaptic neuron, rendering the differential equation linear:

$$\tau_m^{\text{eff}} \dot{V}(t) = -(V(t) - E_L^{\text{eff}}) + R^{\text{eff}} I_p(t), \quad (2.4)$$

with $\tau_m^{\text{eff}} = \tau_m / (E_L + \sum_n E_n)$, $E_L^{\text{eff}} = E_L + \sum_n E_n$, $R^{\text{eff}} = R / (Rg_L + R \sum_n g_n)$. In the following we will drop the superscripts eff for ease of readability, but we should keep in mind that these represent effective parameters in the case of current-based models. This model allows for direct analytical treatment and efficient numerical implementations (Brunel, 2000; Morrison et al., 2007). The simplest version model the current evoked by a presynaptic spike as a Dirac-delta, i.e., the current in the postsynaptic neuron for a spike generated at time t' is given by

$$RI(t) = \tau_m J \delta(t - t' - d), \quad (2.5)$$

where d is modeling a transmission delay arising for example from the propagation of action potentials along the axon, the timescale for molecular processes at chemical synapses and the propagation of postsynaptic potentials along the dendrite. J represents the strength of the connection, which in the case of multiple connections is written as J_{ij} , denoting the connection from neuron j to neuron i . In a network context, J would hence be promoted to a matrix, known as the connectivity matrix. In this case, we can write the input current for a single neuron i receiving input from multiple presynaptic neurons as

$$RI_i(t) = \tau_m \sum_j J_{ij} s_j(t - d) \quad (2.6)$$

where $s_j(t)$ represents the spike train (equation (2.2)) of the j th neuron. To increase the biophysical realism one can introduce more complex shapes of the postsynaptic currents, e.g., exponential or alpha shapes (see, e.g., Fourcaud and Brunel, 2002).

Let us now investigate the behavior of a single leaky-integrate-and-fire neuron in some more detail. In the absence of external input, the membrane potential is at rest. Upon injection of a constant current we see the characteristic charging curve of a capacitor up to a stationary value of RI_e , where I_e denotes the amplitude of the external current. If the stationary value is below the threshold potential V_{th} , the neuron settles in this new stationary state (figure 2.9 A). However, if we apply a strong current, the membrane potential reaches the firing threshold, is reset to the reset potential V_{reset} and the neuron emits a spike. Since the spike generation is not part of the dynamical equation governing the membrane potential dynamics but only implemented as a supplemental condition, it is not visible in the membrane potential recording (figure 2.9). In response to a constant current, the spike train generated by the neuron (spikes times are identical to the time points at which the membrane

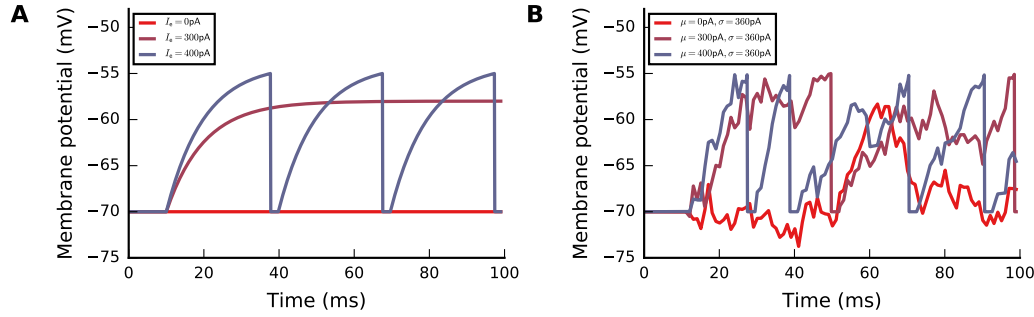


FIGURE 2.9: Leaky-integrate-and-fire neuron models. Response of integrate-and-fire neurons to (A) constant current input and (B) white noise input. The input is switched off for the first 10 ms.

potential is reset) is regular (Mainen and Sejnowski, 1995a). While neurons recorded *in vitro*, i.e., in an isolated slice of tissue, show very similar behavior as our model neuron in response to a constant current, the *in vivo* response of nerve cells looks completely different. *In vivo* recordings show strongly fluctuating membrane potentials and irregular spike trains (Softky and Koch, 1993; Shadlen and Newsome, 1998). These fluctuations arise due to input from other cells in the network, in particular due to a combination of excitatory and inhibitory inputs giving rise to strong fluctuations in the membrane potential and irregular spiking (Shadlen and Newsome, 1998). This behavior of the membrane potential can be mimicked by a model neuron that is stimulated by a Gaussian white noise process with mean μ and standard deviation σ . In this case, threshold crossings occur at irregular intervals and even for currents where the mean input would not suffice to drive the neuron across the threshold. The combination of an average potential below threshold with fluctuations generating spikes is termed the fluctuation-driven regime (figure 2.9). A variation of this model is implemented in the neuromorphic system Spikey (Pfeil et al., 2013b) using analog circuits. We will investigate this more in chapter 6

Neurons in the fluctuation driven regime can, under certain assumptions, be systematically mapped to linear dynamics of continuous variables with additive noise by only requiring identical first and second order statistics of the activity (Grytskyy et al., 2013a). This allows the description of the firing rate of a neuron by an Ornstein-Uhlenbeck process (Gardiner, 1985). In this description, the firing rate can be interpreted either as the spatial average over a population of identical neurons which receive identical synaptic input but additionally uncorrelated noise reflecting our ignorance about the exact state of the individual units. The dynamics of the firing rate in this description is given by

$$\tau \dot{r}_i(t) = -r_i(t) + \sum_j w_{ij} r_j(t) + \xi_i(t), \quad (2.7)$$

with the effective connectivity w_{ij} and Gaussian noise $\xi_i(t) \sim \mathcal{N}(0, \sigma_\xi^2)$. Here the effective connectivity is related to, but not identical to, the physical connectivity of the network. In addition to the physical connectivity, it depends on the working point of the neurons, i.e., the mean and variance of their input. This description simplifies the mathematical analysis of the dynamics of large networks of leaky-integrate-and-fire neurons and is used to describe a number of dynamical phenomena such as low pairwise correlations in recurrent networks (Renart et al., 2010a; Tetzlaff et al., 2012a), the structure of correlation functions (Helias, Tetzlaff, and Diesmann, 2013),

oscillations in layered network models (Bos, Diesmann, and Helias, 2016) or the global stability properties of large-scale models (Schuecker et al., 2017). In addition, it is a popular basis for many neural network models implementing particular high-level computations (Olshausen and Field, 1997; Rao and Ballard, 1999; Orbán et al., 2016).

Before moving on to networks, we discuss one additional, very simple neuron model that will be of major interest when seeking the connection between probabilistic computations and neural substrates. In this model, one distinguishes only two states of a neuron, a high-activity (“on”) and a low activity (“off”) state:

$$s_i \in \{0, 1\}, \quad (2.8)$$

where s_i denotes the state of neuron i at time t . The transition between the two states is modeled probabilistically as probabilistic a function of the total input to the neuron:

$$p(s_i|h) = F_i(h), \quad (2.9)$$

with the instantaneous input field $h_i = \sum_j w_{ij}s_j + b_i$ and some bounded, monotonic activation function F . Here w_{ij} denotes the coupling strength from neuron j to neuron i and b_i a bias that makes the neuron more likely or less likely to fire, depending on its sign. Two different types of activation functions are of particular relevance. One is the deterministic case with

$$F_i(h) = \Theta(h) \quad (2.10)$$

$$= \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{else} \end{cases} \quad (2.11)$$

Transitions between the 0 and 1 states are hence entirely deterministic given a certain value for the input field h . This model is known as the McCulloch-Pitts neuron (McCulloch and Pitts, 1943). It has been very influential in neuroscientific modeling, e.g., describing the emergence of asynchronous activity in large, randomly connected neuronal networks (Vreeswijk and Sompolinsky, 1996a) or the structure of correlations (Helias, Tetzlaff, and Diesmann, 2014a; Dahmen, Bos, and Helias, 2016)). In addition it has been of major influence in the field of artificial intelligence and machine learning, giving rise to so called artificial neural networks, with its earliest representative, the perceptron (Rosenblatt, 1958). The second type of activation function that we will encounter is a logistic function of the input field:

$$F_i(h) = \frac{1}{1 + e^{-\beta h}}, \quad (2.12)$$

leading to stochastic switching between the up and down states (figure 2.10), with an inverse temperature β controlling the amount of stochasticity in the transitions.

In the case of symmetric connectivity and the absence of self-coupling ($w_{ij} = w_{ji}, w_{ii} = 0$), is it possible to derive an analytical expression for the distribution of network states. Since the stationary distribution has Boltzmann form, these models are called Boltzmann machines and we will consider them in more detail in section 2.5. Mathematically, this description is equivalent to the Ising-model, a prominent model in statistical and solid-state physics, which has led to a transfer of tools from physics to neuroscience (e.g., Hertz, Krogh, and Palmer, 1991; Dahmen, Bos, and Helias, 2016).

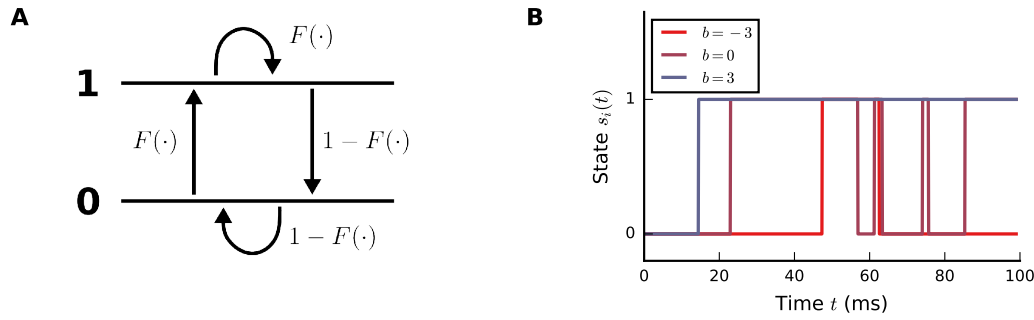


FIGURE 2.10: Binary neuron models. **A** States and transition rates of a binary neuron model. **B** Temporal dynamics of three uncoupled binary neurons with logistic activation function and different biases. Inverse temperature $\beta = 1$, biases $b_i \in [-3, 0, 3]$, inter-update-interval 20 ms.

Apart from the neuron models discussed here, many additional neuron models have been proposed, varying in their biophysical plausibility, mathematical or computational tractability. For more details the reader is referred to standard textbooks (e.g., Dayan and Abbott, 2001; Gerstner and Kistler, 2002). We will now focus on dynamics and function in networks of neuron models.

2.3 Bottom-up network models

Generally one distinguishes two complementary types of network models: those starting from the microscopic constituents and creating models fully based on available experimentally obtained data, typically modeling dynamical features observed in cortical circuits (bottom-up models) and those starting from a certain macroscopic network behavior to investigate possible biophysically plausible implementations of specific functions, such as pattern classification (top-down models). Bottom-up models typically take into account vast amounts of anatomical data to reconstruct neural circuits at the resolution of individual neurons and synapses. Statistical anatomical data suggest that each cortical neuron receives on the order of 10^4 synapses, with about 50% from local neurons (< 1 mm), 30% from within the same area and 20% from remote sources, i.e., other cortical areas or subcortical structures (Abeles, 1991; Braitenberg and Schüz, 1991; Stepanyants et al., 2009). Since the exact connectivity even within a local area is difficult to measure due to the high density of connections, many models assume homogeneous random connectivity within local circuits with an average connection probability of 0.1 consistent with anatomical data (Abeles, 1991). On larger scales however, one needs to take into account the decay of connection probability with distance between two neurons (space constant $\sim 300 \mu\text{m}$) (Hellwig, 2000; Stepanyants et al., 2009; Perin, Berger, and Markram, 2011a; Potjans and Diesmann, 2014; Markov et al., 2014a; Senk et al., 2015). One might interject that the assumption of randomness even for a local circuit is at odds with the idea that computation arises from specific connectivity of large number of cells. There are two possible counter arguments: first, connectivity could be locally random, but globally structured, e.g., with specific connection probabilities between different populations, if computation is organized on a larger spatial scale than the local circuit. Second, specific connectivity needs to be guided by functional interpretations and models, e.g., by training a model on a specific task, leading to a particular network realization. However, in bottom-up models, one is mainly interested in

features that are invariant of the microscopic details of the network realization (so called self-averaging measures), e.g., average firing rates in a population. It can be nevertheless worthwhile in bottom-up models to take into account non-random features of cortical connectivity (Song et al., 2005; Perin, Berger, and Markram, 2011b) and investigate their influence on network dynamics (e.g., Roxin et al., 2011).

One example of a classical bottom-up cortical network model based on binary (Amit and Brunel, 1997; Van Vreeswijk and Sompolinsky, 1998) or current-based leaky integrate-and-fire neuron models (Brunel, 2000) consists of two sparsely and randomly connected populations of excitatory and inhibitory neurons. While recurrent excitatory connections are in the majority, reflecting the 4 : 1 ratio of biological data, inhibitory connections are much stronger, leading to a network state that is stable against small perturbations. The network model is correspondingly referred to as the balanced random network. Despite its simplicity this model captures many aspects of the so called resting state, i.e., the stationary state in absence of a particular stimulus, of local cortical networks:

- low average firing rates (Swadlow, 1988; Kock and Sakmann, 2009; Sakata and Harris, 2009)
- irregular neuron activity (Softky and Koch, 1993; Shadlen and Newsome, 1998; Bernacchia et al., 2011)
- small pairwise correlations in the spiking activity (Ecker et al., 2010)

The spiking version of this network is also able to model the emergence of oscillations in various frequency bands (Brunel, 2000; Helias, Tetzlaff, and Diesmann, 2013).

This network is used as a building block in larger network models, that, for example, take into account the layered structure and corresponding heterogeneous neuron and synapse densities under a 1 mm^2 patch of cortical surface (Potjans and Diesmann, 2014). Studying the dynamical features of such a microcircuit model demonstrates, for example, that the experimentally observed difference between firing rates of excitatory and inhibitory neurons is not necessarily caused by intrinsically different cell properties, but can be accounted for by the specific connection probabilities between different populations alone, while maintaining the general properties of the balanced state. By combining the spiking activity of this network with spatially extended neuron models, one can derive measures that can be directly compared to electrophysiological data, for example electrophysiological recordings (Hagen et al., 2016). Modern large-scale network models, comprising on the order of 10^6 neurons and 10^{10} synapses, generalize this local circuit model by adding long-range interactions of multiple areas with different cell and synapse densities (Schmidt et al., 2016) or introducing distance dependent connectivity in a patch with larger spatial extension (Senk et al., 2015). Due to its simplicity, the two-population model and its derivatives are amenable to analytical statistical descriptions based on the Fokker-Planck formalism (Brunel, 2000). This allows deep insights into the role of specific network parameters on the emergent network dynamics, e.g., how connectivity statistics between and within different layers shape oscillations in different frequency bands (Bos, Diesmann, and Helias, 2016).

Further studies include more biophysical details (Markram et al., 2015), combine dynamics at different spatial and temporal scales (Litwin-Kumar and Doiron, 2012), investigating network stability and emergence of tightly coupled clusters of neurons in the presence of plasticity (Morrison, Aertsen, and Diesmann, 2007b) or modeling large-scale dynamics as found in fMRI data (Bressler and Menon, 2010; Deco,

Senden, and Jirsa, 2012; Hansen et al., 2015). However, in contrast to other physical systems, the questions we would like to address in neuronal networks have a different quality: in addition to the dynamical features we are interested in understanding how our brain implements certain high-level cognitive functions, such as pattern recognition, memory or planning. This is the arena of functional neural network models that we will consider next.

2.4 Top-down network models

Top-down models aim at reproducing system-level behavior while taking into account biophysical constraints and can guide bottom-up models, thereby bridging intermediate scales. By focusing on the functional aspect of neural networks, leaving aside many details of the structure of biological networks, they can provide constraints that are missing or difficult to obtain from biological data and allow us to extract fundamental principles of the organization of dynamics of cortical circuits required to produce certain macroscopic behavior. Trying to understand brain function purely from a bottom-up perspective is similar to trying to replicate a fully functional computer just by investigating the structure of connections between transistors in an existing one and replicating basic dynamical effects without a clear idea of what the full system is supposed to do (Jonas and Kording, 2017).

The main focus of top-down models is not on hard-coded solutions, i.e., networks that were manually wired to perform a certain task, but on networks that learn from labeled data (supervised learning), unlabeled data (unsupervised learning) or reward and punishment (reinforcement learning) with biophysically plausible plasticity rules. One important constraint of these learning rules is locality of information: all variables involved in a particular rule should (at least in principle) be available at the location of the physical synapse. This is at odds with some of the most popular learning rules in machine learning, e.g., backpropagation of errors in supervised learning (Rumelhart, Hinton, and Williams, 1986).

Typical tasks that are considered in functional network models include function learning/approximation (Eliasmith and Anderson, 2004), pattern classification (Gütig and Sompolinsky, 2006), completion and generation (Hinton, Sejnowski, and Ackley, 1984a), memory (Hopfield, 1982a), time series prediction (Brea et al., 2016) or navigation (Samsonovich and McNaughton, 1997; Conklin and Eliasmith, 2005). A plethora of different functional network models exist targeting different applications, with the earliest from Rosenblatt (1958), the so called perceptron that was able to learn to distinguish visual stimuli. The first functional fully recurrent network models were attractor neural networks, introduced by Hopfield (1982a). These can store memories that are accessible by presenting parts of a memory to the network, forming a type of content-addressable memory, and can be extended to perform more complex temporal tasks (Amit, 1988). Recent versions of these types of network consider networks of spiking neurons arranged in several cortical columns (Lundqvist et al., 2006a; Lundqvist, Compte, and Lansner, 2010), replicating typical signatures of cortical dynamics while maintaining their computational power. While many successful functional network models are based on continuously interacting units with graded responses (e.g., Camperi and Wang, 1998), recent years have seen a surge in interest in functional spiking network models (see Abbott, DePasquale, and Memmesheimer, 2016, for a review) from which many researchers expect additional computational power (Maass, 1997) and a more direct comparison to

biological data. Recent developments focus on translating successful machine learning approaches like backpropagation to biophysically plausible spiking networks for pattern recognition tasks (e.g., Lee, Delbruck, and Pfeiffer, 2016). Another successful computing paradigm in spiking networks is reservoir computing that uses the fading memory inherent in recurrently connected neuronal networks to implement computations on temporal data (Maass, Natschläger, and Markram, 2002). The original approach has been extended, for example to enable networks to learn complex motor control tasks (Sussillo and Abbott, 2009a).

A very successful avenue of research was the extension of Hopfield networks to stochastic single-unit dynamics that prevents the networks from running into the closest fixed point, thereby improving pattern retrieval performance in difficult scenarios and allowing the network to autonomously generate patterns in the absence of inputs (Hinton, Sejnowski, and Ackley, 1984a). We will discuss this model in detail in the next section.

2.5 Probabilistic computations in neural substrates

As argued in section 1.1 probability theory provides a normative framework for plausible reasoning under uncertainty. However, due to the immense computational effort inherent in real-world tasks, humans and other organisms must implement ingenious shortcuts for performing approximate inference (section 1.2). Mechanistic models that demonstrate how neuronal networks can implement the required computations on the basis of individual neurons and synapses are hence of particular relevance. These allow us to connect macroscopic theories about the behavior of intelligent organisms with a mechanistic understanding of how the interaction of the microscopic constituents can give rise to such behavior. In particular they can be used to make concrete predictions about dynamics of neuronal networks performing inference tasks that can be experimentally verified. Such circuit-level models need to address the question of how probability distributions are represented and transformed in a way that is consistent with current biophysical evidence. Two general classes of models are currently of particular interest: spatial representations of distributions such as probabilistic population codes (PPCs) (Ma et al., 2006) and temporal representations via sampling-based approaches such as neural sampling (Buesing et al., 2011).

PPCs were inspired by the observed variability in population responses to fixed stimuli that naturally leads the representation of (probability) distributions via activity statistics. They exploit the observed Poisson-like variability of cortical neurons to represent distributions over possible stimuli by explicitly encoding the most likely stimulus and its uncertainty in the population activity (Ma et al., 2006; Beck et al., 2008). The most likely stimulus and its uncertainty can be interpreted as parameters of some well-defined probability distribution from the exponential family. Given a certain stimulus, each neuron in the population exhibits stochastic responses proportional to its sensitivity, i.e., tuning, to the given stimulus. With each neuron of a population preferring a slightly different stimulus, the most active unit(s) represent(s) the best guess for the stimulus, while the relative contribution of the most active unit(s) to the total population activity is inversely proportional to the population's certainty about this estimate. A strong argument for this type of representation is that computation on these distributions, e.g., calculating posteriors, are implemented in a biologically plausible way via linear combinations of population activities. This allows a system to represent an approximation to the full posterior

at any point in time. PPCs were extended to incorporate models of evidence integration and action selection that generate activity similar to LIP neuron responses recorded from monkey in continuous decision making tasks (Beck et al., 2008). To seek the link to cognitive phenomena like perceptual multi-stability that are absent in a purely spatial representation, recent versions of PPCs include temporal extensions, for example by sampling from the posterior via attractor network dynamics (Moreno-Bote, Knill, and Pouget, 2011).

Another prominent approach for implementing approximate inference in neural substrates is via Markov chain Monte Carlo (MCMC) sampling (section 1.3). Typical characteristics of MCMC sampling can be related to cognitive phenomena (see section 1.3), and the stochasticity intrinsic to the sample-based representations is consistent with the observed variability of neural responses (Hoyer and Hyvarinen, 2003; Lee and Mumford, 2003). One of the earliest models for representation and computations with probability distributions in neuronal networks based on MCMC methods were Boltzmann machines (Hinton, Sejnowski, and Ackley, 1984a; Ackley, Hinton, and Sejnowski, 1985), which also have been proposed to solve problems in image processing (Geman and Geman, 1984a). In the following we discuss a few general properties of this network model. Boltzmann machines are networks of symmetrically connected, stochastic binary units with logistic activation functions (cf. section 2.2.2) that implement a certain type of Gibbs sampling. The stationary distribution of the Markov chain, or of networks states in the current context, is given by the Boltzmann distribution (Hinton, Sejnowski, and Ackley, 1984a)

$$p(\mathbf{s}) = \frac{1}{Z} e^{-E(\mathbf{s})}, \quad (2.13)$$

where $E(\mathbf{s})$ represents the energy of the state $\mathbf{s}$, given by

$$E(\mathbf{s}) = -\frac{1}{2} \mathbf{s}^T W \mathbf{s} - \mathbf{b}^T \mathbf{s}, \quad (2.14)$$

W denotes the weight matrix and $\mathbf{b}$ the bias vector. According to the procedure discussed in section 1.3, only a single unit is updated in each step, and each time all units have been updated, the network state is, after an initial burn-in period, a sample from the stationary distribution of the Markov chain. Computations on the represented distribution such as marginalization or conditioning can easily be performed via sampling. To compute the marginal probability over the subset of the first k random variables (RVs), here denoted by $p(\mathbf{s}_k)$ with $\mathbf{s}_k = (s_1, \dots, s_k)^T$, from the full distribution $p(\mathbf{s})$ over n variables one needs to integrate over the remaining $n - k$ variables:

$$p(\mathbf{s}_k) = \int ds_{k+1} \dots ds_n p(\mathbf{s}).$$

A sampling method provides samples distributed according to p , and allows us to perform integrals of arbitrary observables via the empirical average over samples (equation (1.4)). We hence rewrite the marginalization integral over $n - k$ dimensions as an n dimensional integral by inserting Dirac delta distributions $\delta(\cdot)$, and can then

approximate the full integral via a sum over samples from p :

$$p(\mathbf{s}_k) = \int ds'_{k+1} \dots ds'_n p(\mathbf{s}_k, \mathbf{s}'_{\setminus k}) \quad (2.15)$$

$$= \int ds'_1 \dots ds'_k ds'_{k+1} \dots ds'_n p(\mathbf{s}') \delta(s'_1 - s_1) \dots \delta(s'_k - s_k) \quad (2.16)$$

$$\approx \frac{1}{N} \sum_{\substack{i=1 \\ \mathbf{s}' \sim p}}^N \delta(\mathbf{s}'_k - \mathbf{s}_k). \quad (2.17)$$

To calculate marginals we hence generate samples from the full distribution but only observe a subset of the variables of each sample corresponding to the RVs we are interested in, which is much easier than performing the high-dimensional integral over $n - k$ variables. Computing the conditional distribution via sampling, e.g., $p(\mathbf{s}_k | \mathbf{s}_{\setminus k})$ is not as straightforward in general, but turns out to be simple in the case of Boltzmann machines. We discuss this in detail in chapter 7.

To successfully perform inference on partial and ambiguous information, we require prior expectations that need to be learned from the statistical properties of the environment. In the following we will discuss an approach to learning in Boltzmann machines that is guided by mathematical convenience and machine learning ideas rather than biophysical considerations. Since an analytical form of the stationary distribution of networks states p is known, one can derive learning algorithms that shape this distribution arbitrarily, for example to approximate some given data distribution q . One can reduce the deviation between the stationary distribution of network states p and the data distribution q by adjusting weights and biases for example by (approximately) minimizing the Kullback-Leibler divergence (D_{KL}) between p and q . For a discrete distribution as in this case, the D_{KL} is defined as:

$$D_{\text{KL}}(p, q) = \sum_i p_i \log \left(\frac{p_i}{q_i} \right) \quad (2.18)$$

Taking the derivative of this objective function with respect to the weights and biases, and using these derivatives to update the parameters, i.e., performing gradient decent ($\Delta\theta \sim -\partial_\theta D_{\text{KL}}$ for parameter θ) on the objective function, leads to the following update for the parameters of the model (Ackley, Hinton, and Sejnowski, 1985):

$$\Delta w_{ij} \sim \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{free}}, \quad (2.19)$$

$$\Delta b_i \sim \langle s_i \rangle_{\text{data}} - \langle s_i \rangle_{\text{free}}, \quad (2.20)$$

effectively matching the first two moments of the two distributions. Here $\langle \rangle_{\text{data}}$ denotes the average over the training data and $\langle \rangle_{\text{free}}$ denotes the average over the stationary distribution of the Markov chain. One issue with this direct approach is that calculating the second term is computationally expensive since the Markov chain has to have reached equilibrium, and many samples are necessary to achieve a good estimate of the gradient to reduce the variance of parameter updates. Hinton (2002) proposed a shortcut that significantly accelerates learning, called contrastive divergence (CD), does not require multiple samples from the stationary distribution. It rather considers only k updates of the network states (CD- k) and additionally considers only a single data sample for the first term, leading to stochastic gradient

descent. The above formulas are accordingly replaced by

$$\Delta w_{ij} \sim s_i^d s_j^d - s_i^k s_j^k, \quad (2.21)$$

$$\Delta b_i \sim s_i^d - s_i^k, \quad (2.22)$$

where s^d denotes a single data vector and s^k denotes the network state after the k th update. This method works remarkably well even for just a single update of the network state, i.e., CD-1. To reduce the variance of each weight update, multiple parameter updates (Δw_{ij} , Δb_i) are typically collected and averaged before performing an actual update of the parameters, leading to so called mini-batches.

Two features are particularly interesting about this learning rule: first, it uses only local information, i.e., to update the weight between neuron i and j , we only need to observe the activities of neuron i and j . Second, it is an unsupervised learning algorithm, i.e., it is able to detect structure in data without requiring us to provide explicit labels for each data sample. However, since we only adjust the parameters of a Boltzmann distribution, the network will only be able to represent Boltzmann distributions. This is a significant drawback since the Boltzmann distribution only captures up to pairwise dependencies between random variables, which typically is not sufficient for natural tasks and can for example not account for the “explaining away” effect, in which observation of one cause for an event modifies the belief in other causes that could entail this event (Wellman and Henrion, 1993). To tackle this problem, extensions of the original model were proposed that introduce so called hidden units, i.e., neurons that are not directly exposed to the input, in contrast to the visible units. To simplify the training, connections within a layer are removed and neurons are only connected to units in the other layer (restricted Boltzmann machines, RBMs Hinton and Salakhutdinov (2006a)). While the full distribution over hidden and visible units in a RBM is still a Boltzmann distribution, the marginal distribution over the visible units can now take arbitrary shapes with their representational power only limited by the number of hidden units (Le Roux and Bengio, 2008). In combination with being able to train with unlabeled data, this feature makes the algorithm interesting for large-scale problems in machine learning (Hinton and Salakhutdinov, 2006a; Hinton, Osindero, and Teh, 2006).

To illustrate the learning procedure we train a fully visible Boltzmann machine, i.e., one without hidden units, on a probabilistic variant of the XOR problem. We consider a target distribution p over two binary RVs, in which the states 01 and 10 are much more likely than the states 00 and 11. To be concrete, in vectorial notation p takes the values: $\mathbf{p}(x) = (0.01, 0.49, 0.49, 0.01)^T$, with $x = (00, 01, 10, 11)^T$. We train a fully visible BM on sample data generated from this distribution using mini batches of size three, a linearly decreasing learning rate, momentum and weight decay (Hinton, 2010). After about 10^6 training episodes the network has internalized the data distribution, and the stationary distribution of network states that the network generates in absence of any input is almost identical to the data distribution (figure 2.11).

This demonstrates an additional feature of these models: by storing the full probability distribution of the data they can be used as discriminative (sampling from the conditional $p(x|y)$) as well as generative models, producing new inputs with similar statistics as the training data while no input is provided (sampling from the joint $p(x, y)$). This particular dynamical signature provides a link to experimental data: spontaneous generation of patterns of activity that have remarkable resemblance to stimulus evoked activity has been reported in experimental data (Kenet et al., 2003; Ringach, 2009). Based on these observations a recent study compared statistics of

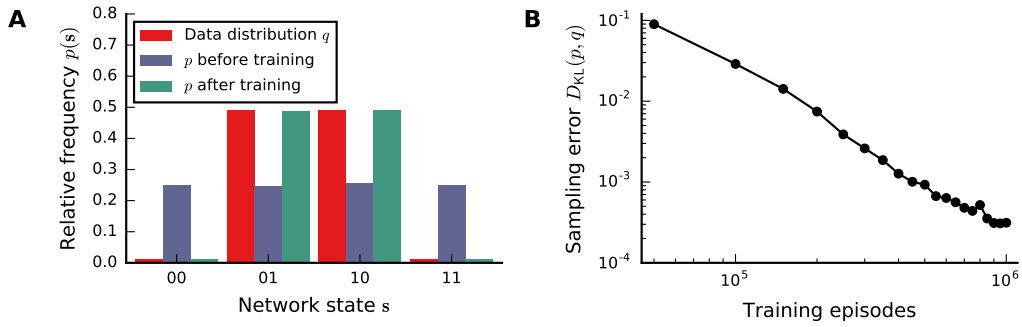


FIGURE 2.11: Learning in Boltzmann machines. **A** Target distribution q (red) and empirical distribution of network states p before (blue) and after training (green). **B** Sampling error $D_{KL}(p, q)$ as a function of training episodes.

stimulus evoked and spontaneous activity from multiunit recordings in the visual cortex of ferrets over the course of development. The authors found an increasing resemblance between spontaneous and stimulus evoked activity for natural stimuli, but not for artificial stimuli, suggesting an adaptation of an internal generative model to the features of the environment (Berkes et al., 2011).

Recent developments in the field have focused on including more biophysically plausible aspects in sample-based approaches, for example including communication between units via spikes and refractory effects (“neural sampling”) (Buesing et al., 2011; Pecevski, Buesing, and Maass, 2011). Two problems arise in these particular methods through identifying a random variable with a particular unit in the network: a one-to-one mapping is not a very robust representation and it is difficult to represent time varying distributions since a non-negligible time is needed to collect enough samples for a good approximation. Both of these issues can be addressed by representing a single RV with an ensemble of neurons (Savin and Deneve, 2014; Legenstein and Maass, 2014). Moving further towards biologically plausible dynamics and possible neuromorphic hardware implementations (see below) researchers have recently considered deterministic leaky-integrate-and-fire models with external sources of stochasticity to replace the intrinsically stochastic binary units in previous approaches (Probst et al., 2015; Petrovici et al., 2016). Other extensions of the Gibbs sampling method, so called Hamiltonian sampling, can improve the convergence of the Monte Carlo chain. When interpreted as the dynamics of a neuronal network model, in particular with the distinction of excitatory and inhibitory populations, it can explain certain experimental observations, like oscillations in the population activity in response to constant input and strong transients after stimulus onset (Aitchison and Lengyel, 2016). One important open question regarding approximate inference in neural substrates that is neither solved via PPCs nor by neural sampling is how the internal probabilistic model is learned and updated over the lifetime of an organism.

A central issue for a biophysically plausible theory of approximate inference via sampling is the origin of the stochasticity required for the sampling process. The assumption of a particular form of intrinsic stochasticity made in many current models, e.g., neural sampling, is at odds with the observed deterministic behavior of single, isolated neurons in vitro (Holt et al., 1996). Since we rely on the description of nervous systems in terms of simple phenomenological models of neurons and synapses, we ignore a huge number of degrees of freedom of neural systems that potentially force us to consider stochastic dynamics on the effective level of description, in the neuroscience literature known as trial-to-trial variability (see, e.g., Buice

and Chow, 2011). At the level of individual neurons, the experimental literature reports a variety of noise sources that can contribute to the observed trial-to-trial variability of neuronal recordings (see, e.g., Faisal, Selen, and Wolpert, 2008a, for a review). Channel noise for example describes the stochastic opening and closing of ion channels in the neuronal membrane. Its contribution to the total variability of the potential *in vivo* is rather small and significantly influences action potential generation only when the potential is already close to threshold causing jitter in spike timings (Faisal and Laughlin, 2007). Some of the main sources of variability arise at chemical synapses. First, even in the absence of presynaptic activity, neurotransmitter can be released in the synaptic cleft leading to a spontaneous change of the postsynaptic potential. While these events can have a significant amplitude (~ 0.1 mV), they are quite infrequent and hence contribute only little to the variability *in vivo*. Second, in the presence of presynaptic activity, synaptic transmission can sometimes fail completely, or lead to postsynaptic potentials of varying amplitude (Faisal, Selen, and Wolpert, 2008a). The case of failure is quite prominent, with only 20% to 80% of all generated action potentials on the presynaptic side actually generating a postsynaptic response (Branco and Staras, 2009). Note that this reliability is affected by neuromodulators and hence potentially serves a computational purpose rather than reflecting a flaw of the molecular machinery (Faisal, Selen, and Wolpert, 2008a). Third, one can interpret parts of the large amount of input that a neuron receives as additive background noise if we assume that not all inputs are contributing information that is currently relevant for the local computation. It is possible that many of the weak synapses observed in anatomical data serve exactly this purpose (see, however, Goulas, Schaefer, and Margulies, 2015). In general it is very difficult to tease apart the different contributions in experimental data, and this is a concrete point where mechanistic models at cellular resolution guided by functional considerations can provide necessary insights.

Chapter 3

Simulation technology

Numerical simulation has become one of the corner stones of modern science to investigate systems with many degrees of freedom and strong, non-linear interactions. In the natural and engineering sciences, simulations are applied to problems from nuclear physics (Borsanyi et al., 2010), to meteorology (Zängl et al., 2015) and energy systems (Hilpert et al., 2017), and to the largest structures of the universe (Giacomazzo et al., 2012). In neuroscience, computational approaches have a long history and recent advances in software (e.g., Kunkel et al., 2014; Lytton et al., 2016) and hardware tools (Furber, 2016) allow for the routine simulation of neuroscientific models of unprecedented complexity. Representing the entire spectrum of interests of researchers in computational neuroscience, a range of simulators target various scales of description. These range from the molecular level, for example to understand the interaction of certain ion channels with particular chemicals via molecular dynamic simulations (Wils and De Schutter, 2009) to coarse-grained approaches that consider the interaction of different neuronal populations or even brain areas as the fundamental level (de Kamps et al., 2008; Sanz Leon et al., 2013). At the intermediate level of individual nerve cells tools are available for the simulation of small-scale networks of complex neuron models that take into account the spatial extension of nerve cells in fine detail (Carnevale and Hines, 2006; Bower and Beeman, 2007) and complex medium-scale to large-scale networks of simple point-neuron models (Gewaltig and Diesmann, 2007; Goodman and Brette, 2013). In addition to these simulators, the recent decade has seen rapid development in the field of neuromorphic engineering, the idea of emulating neuronal systems in dedicated very-large-scale-integrated (VLSI) circuits (Mead, 1990) instead of relying on software running on traditional microprocessors.

One important consideration in designing simulation tools arising from the interdisciplinary character of the field is the ease of accessibility: Domain experts, i.e., (computational) neuroscientists should be able to exploit the full potential of the available systems without the help of an expert. In addition it should be easy for users to implement new models not included in the simulator by default, since model development is a key-aspect of current neuroscientific research. Users should be able to define a simulation using a level of abstraction familiar to them, for example in terms of neurons or populations that can be connected with arbitrary connectivity rules – advanced knowledge of the internals of the simulation backend should not be a requirement. One exemplary project that demonstrates this approach is PyNN (Davison et al., 2008a), allowing users to define a simulation on the level of neurons and synapses in simple, procedural scripts and later run the simulation using different backends, software simulators and hardware platforms. This abstraction, hiding the complex implementation details from users, allows an important separation of infrastructure development and neuroscientific research. With the growing complexity of simulation code sustainability and maintenance of software

become important issues and high-quality, systematic software development practices become essential, which can and should not be expected from each individual researcher in computational neuroscience. As the complexity of the neuroscientific models increase, availability of well-tested and maintained simulation systems is essential to support correctness and reproducibility of results of simulation studies. This is a prerequisite for other researchers to be able to extend and build upon previous work, which is barely possible with in-house, custom-made software that was tailored to the respective projects. One issue arising especially when developing functional network models is the systematic generation of appropriate input to the network to evaluate, for example, the performance of specific learning rules. Chapter 9 introduces a toolchain that allows researchers to supply a neural network simulation with realistic input for reinforcement-learning scenarios directly from well-established toolkits thus avoiding the manual implementation of training tasks.

If we combine the average number of connections (10^4) with the average connection probability (0.1) measured in biological data (Abeles, 1991), we need to consider a network of at least 10^5 neurons to satisfy both constraints at the same time. A popular approach to mitigate the computational complexity is to consider down-scaled networks in which both the number of neurons and the number of connections are reduced, with other parameters such as the synaptic weight adjusted to compensate for this modification. However, down-scaling of networks is severely limited even if one would like to conserve only the first two moments of the single unit activities (Albada, Helias, and Diesmann, 2015). Since pairwise correlations have a significant impact on electrophysiological measures like the local-field potential (LFP) (Lindén et al., 2011), down-scaled models can lead to wrong predictions and full-scale models should be used to compare simulated to experimental data (Hagen et al., 2016). Note that beyond 10^5 units, the number of connections grows linearly with network size such that the network becomes increasingly sparse (Lansner and Diesmann, 2012a).

Even for models of mammalian cortex that contain $\sim 10^5$ neurons about 50% of the projections to neurons are unaccounted for since they arise from sources outside this local area (Stepanyants et al., 2009). These are usually replaced by input generated via independent Poisson processes and hence cannot account for phenomena involving the interaction of multiple different areas, for example slow fluctuations observed in resting-state fMRI (Deco and Jirsa, 2012). To achieve a self-consistent description we need to consider large-scale models that include remote areas and take into account, for example, the long transmission delay between distant areas (Schmidt et al., 2016). Models containing about 10^6 single-compartment neurons represent the largest models currently considered (Senk et al., 2015; Schmidt et al., 2016). Based on this observation, one could argue, that there is no reason to further develop simulation technology capable of simulating networks on the scale of 10^9 or more individual neurons with a realistic density of connections. However, one should keep in mind that once the fundamental features of a model are defined and well understood, scaled-up versions are often needed to investigate phenomena that are absent in the original model. One example is the use of the popular balanced random network model with about 10^4 neurons as a basic building block in current large-scale models with more than 10^6 neurons. If we are serious about our hypothesis that individual neurons and synapses are relevant for macroscopic function (cf. section 2.1), we need to be prepared to simulate networks of 10^9 neurons for neuroscientific purposes in the near future and hence willing to allocate resources for the development of appropriate simulation technology.

3.1 Computational challenges of large-scale neural-network simulations

In simulations of networks with simple single units and complex connectivity with biologically plausible density (see section 2.2.2), the main computational challenge arises from the high density of connectivity and the corresponding communication, while the computational costs for propagating single-cell dynamics are negligible. With each neuron receiving on the order of 10^4 synapses and 10^{10} neurons in the human cortex, we need to represent about 10^{14} individual connections to simulate a network of realistic size at a resolution of individual synapses. Assuming that each connection is represented by 16B¹, we require approximately 1.42PB just to store all synapse objects, not accounting for the memory consumption of neurons or supporting data structures. The contribution of neurons can be safely ignored in this calculation, since their number is three to four orders of magnitude smaller for a realistic density of synapses, and hence even models with hundreds of state variables consume a negligible amount of memory compared to the memory usage of connections (Morrison et al., 2005). The total memory usage of synapses calculated above corresponds to about three times the main memory of JUQUEEN, the current supercomputer in the Research Centre Jülich, ranked 21st in the top500². This demonstrates that our main focus in developing simulation technology for large-scale networks needs to be on efficient representations of connectivity. In particular we need to exploit high-performance-computing (HPC) systems that enable the distribution of network simulations over many compute nodes. These systems are basically a large number of workstations equipped with fast interconnects that allow parallel processes running on each node to exchange data efficiently. In neuromorphic systems employing custom circuits for individual synapses, the high density of connections leads to most of the chip area being occupied by synaptic circuitry, with neurons requiring only a minuscule fraction of the total chip (Pfeil et al., 2013b). To reduce the memory requirements, which are in particular expensive in neuromorphic hardware, it was hence proposed to reduce the bit precision for synaptic weights in neuromorphic applications (Pfeil et al., 2012; Stomatias et al., 2015a).

One aspect of neuronal networks that makes the use of distributed architectures challenging is the presence of many long-range connections. In contrast to other physical systems, e.g., fluids, where the interaction between the simulated entities is local in space³ allowing the use of finite-element methods that can conveniently map the spatial structure of the physical system to the network structure of a particular HPC system, neuronal simulations need to take into account the non-negligible probability of a neuron targeting any other neuron in the network (Braitenberg and Schüz, 1991; Stepanyants et al., 2009). This makes it difficult to map the spatial structure of neuronal networks to the (typically toroidal) communication-network architecture on modern supercomputers, requiring neuronal simulators to implement efficient communication schemes that enable fast simulations despite the necessity of global communication across all compute nodes. Here, the spike-based communication between biological neurons simplifies matters: since individual units are not instantaneously coupled, neurons evolve independently over the duration of the minimal delay in the network, allowing simulators to use separate step sizes for the

¹For example 8B for weight in double precision, 8B for pointer to target on a 64 bit architecture.

²July 2017, see <https://www.top500.org/list/2017/06/>

³Potential $\sim 1/r$ if mediated via Coulomb interaction

propagation of neuron dynamics and for communicating spikes. One can hence employ efficient communication algorithms while maintaining an accurate propagation of state variables (Morrison et al., 2005; Hines and Carnevale, 2008).

Additional computational challenges in the simulation of neuronal networks include the accurate and fast numerics for neuronal state updates (Morrison et al., 2007) and learning rules, efficient delivery of spikes via synapses, the continuous interaction introduced by gap junctions (Hahne et al., 2015) or rate models (Hahne et al., 2017) and rewiring of neurons during a simulation (Diaz-Pier et al., 2016).

3.2 Software tools

The systematic development of neuronal simulation tools has been gaining traction only over the recent decade (Brette et al., 2007; Djurfeldt, Ekeberg, and Lansner, 2008), with researchers previously employing custom code that was, in many cases, only properly curated for about as long as a specific research project lasted. With the complexity of the models ever increasing, the community gradually realizes that well-tested high-performance simulation software is a required basis for sustainable progress. Software tools employing off-the-shelf computing hardware provide immense flexibility in neuron, synapse and network models, only limited by the researchers imagination. Implementation of new neuron or synapse models can be done, for example, by defining dynamics via differential equations in standard mathematical notation (Goodman and Brette, 2009), domain specific languages like NESTML (Plotnikov et al., 2016a) or NEUROML (Gleeson et al., 2010) or via C++ code (Gewaltig and Diesmann, 2007).

Current simulation engines support the routine simulation of networks of local cortical populations with $\sim 10^5$ individual neurons connected by 10^9 synapses on a typical workstation. These simulators use a hybrid of time- and event-driven approaches for the propagation of network dynamics. Individual neurons are updated on a time grid due to the large input count of individual neurons: with about 10^4 synapses transmitting events at about 1 Hz, each neuron receives an event every 0.1 ms, such that a time-driven approach with this resolution is at least as efficient as an event-driven method (Morrison et al., 2005). With each synapse transmitting only a single event per second, synapses are more efficiently updated in an event-driven fashion, reducing the computational load massively (see Morrison et al., 2005, for a detailed discussion). Despite the event-driven update, spike delivery via synapses to neurons is a possible bottleneck in simulations, as it requires constantly accessing random addresses in memory causing significant amounts of data being transferred between caches and main memory.

Due to the high numbers of connections, memory consumption is a major concern for large-scale simulations, ultimately limiting the maximal network size that can be investigated. If we do not want to lump together synaptic state variables, i.e., allow for heterogeneous weights that can be individually modified by plasticity rules, we need to represent each synapse as an individual object in memory. To escape the severe memory limitations of single machines parallel simulation codes were developed that support the distributed⁴ simulation of neuronal networks (Morrison et al., 2005; Migliore et al., 2006). Parallelization across different machines requires the design of suitable data structures and algorithms that handle the exchange of data, for example spikes, between multiple processes. Modern

⁴Here we use *distributed* to refer to concurrent processes without access to shared memory, exchanging data, for example, via MPI.

simulation tools typically use a so called address event representation (AER, Boahen, 2000) scheme in which a single spike is represented by a global identifier for the source or target neuron, while time is implicitly encoded in the arrival of the data. Synapse parameters such as the synaptic weight are typically stored on the same node as the target neuron (see Lansner and Diesmann, 2012a, for a review). In addition to efficient communication patterns, storing synapses on the processes of the target neurons allows for massively parallel network construction which, even if fully parallelized, can make up for a significant amount of wallclock time (Kunkel et al., 2014). The communication of spikes typically relies on collective communication routines from the Message Passing Interface (MPI) library (Message Passing Interface Forum, 2009), the de facto standard for communication between processes in scientific HPC applications. For up to 10^4 distributed processes, a single neuron with an average out-degree of 10^4 will have targets on any machine, making the `MPI_Allgather` communication routine an optimal choice (Migliore et al., 2006; Hines, Kumar, and Schürmann, 2011).

Since the first distributed simulation codes for neuronal networks, several advances in data structures and algorithms have pushed the boundaries of the largest network that is possible to simulate on current top-tier HPC systems to about 10^9 neurons, corresponding to about 10% of human cortex. These simulations consume the whole main memory of contemporary supercomputers (Helias et al., 2012; Kunkel et al., 2014) and take about 40 minutes to simulate one second of biological realtime. For simulations at this scale it is of primary importance that the memory usage per compute node should not increase with the total number of neurons in the network or the number of MPI processes. Even just a single bit per neuron in the network leads to a memory consumption of more than 100MB for networks of more than 10^9 neurons, a non-negligible part of the local memory available on each compute node on current HPC systems. This demonstrates that to guide the development of new data structures and algorithms for large-scale simulations, runtime and memory-consumption models are of huge importance as they reveal potential bottlenecks without running unnecessary, resource-intensive simulations on HPC systems (Kunkel et al., 2012a; Kunkel et al., 2012b). In chapter 8 we present a new simulation kernel that employs a two-tier connection infrastructure and directed communication to remove scalability bottlenecks in previous simulation technology.

Current HPC architectures support additional levels of parallelism on the node level via threads, and on the core level via vector instructions. Many modern systems include dedicated accelerators such as general-purpose graphics processing units (GPGPUs) and field-programmable gate arrays (FPGAs). To make efficient use of HPC facilities, simulators need to support these multiple levels of parallelism (see, e.g., Plesser et al., 2007). Threading on shared-memory machines can be integrated into existing simulation code in a straight-forward fashion, but with some care about shared-memory access and is therefore supported by most neuronal networks simulators. However, some care needs to be taken to make sure threads are not unnecessarily slowed down due to non-thread-aware memory allocators (Ippen et al., 2017). On the finer scale, vector operations in modern processors and GPGPUs require appropriate memory layouts of the most fundamental data structures in the simulator and hence are more difficult to exploit, but can significantly boost the simulation speed (Nageswaran et al., 2009). One particular issue arising in GPGPU implementations is the memory-access bottleneck. As discussed above, spike delivery via individual synapses leads to a large amount of random reads and writes of small amounts of data from and to memory – therefore the latency of accessing data from main memory in the graphics processor potentially limits performance gains

from accelerators.

3.3 Neuromorphic hardware

Neuromorphic hardware systems present a significant departure from the traditional von-Neumann computing paradigm. Inspired by “the awesome power of collective systems to solve problems that are totally intractable by traditional computing techniques”, Mead (1990) advertised the idea of using analog VLSI circuits to mimic basic neuro-biological information processing principles. Compared to simulations on traditional computing architectures, the neuromorphic approach promises to deliver a significant speedup accompanied by a massive reduction in energy consumption and an inherent fault tolerance. This makes them not only interesting for neuroscientific modeling but also for the fields of machine learning, artificial intelligence, and robotics.

Current (super)computers consume massive amounts of energy for tasks like pattern recognition in natural environments, predictions from sparse data, navigation, and motor control, which our brains perform with incredible ease on a slim energy budget, despite its comparatively large and slow processing units and seemingly unreliable components. Our current computing architectures differ fundamentally from neurobiological systems, although they might share superficial similarities such as all-or-none events. Built on the principles of the universal Turing machine (Turing, 1936) the von-Neumann computing architecture (Von Neumann, 1945) has a clear separation of processing units and memory, requiring large amount of data to be moved around for memory-intensive applications. In addition such an architecture follows a serial order of operations, executed deterministically with very little tolerance to failure of individual components. These differences are the main reasons why simulations of the massively parallel dynamics of neuronal networks on current computing systems consume orders of magnitude more power than their biological counterparts, while running at a fraction of the speed. For example, to simulate 1s of activity in a network with an equivalent number of neurons and synapses as 10% of human cortex on the JUQUEEN supercomputer takes about 40minutes (Kunkel et al., 2014) consuming about 2 MW of power, compared to the ~ 10 W our brains require (Herculano-Houzel, 2011). By constructing computing systems following the principles observed in biology, one hopes to overcome this huge gap by creating machines with tight integration of memory and processing units, massive parallelism and inherent tolerance to noise and device failure. This is of particular relevance in areas where either energy (e.g., robotic applications) or time (e.g., neuroscientific models of learning and development) is a limiting factor for progress. Partly we can observe the trend to massive parallelism also in modern microprocessor architectures, for example the inclusion of hundreds of threads in a single CPU and support for vector operations (see above). This development is partly fueled by Moore’s law coming to a halt due to technology reaching the physical limits of scalability (Lloyd, 2000; Cavin, Lugli, and Zhirnov, 2012).

Several different approaches to neuromorphic computing exist today, partly adhering to Mead’s original idea of using analog VLSI circuits to emulate neurons and synapses (e.g. Benjamin et al., 2014; Schemmel et al., 2010a) and partly retreating to the fully digital domain with a focus on massive parallelism (e.g., Furber et al., 2014; Merolla et al., 2014). Most of the current systems aim at an abstraction on the level of simple neuron and synapse models and can be distinguished by particular design trade offs, in particular between energy efficiency and flexibility.

One of the main strengths of analog approaches is that important mathematical functions such as exponentials, that are expensive to compute in software, result from fundamental physical laws (Mead, 1989), which can be exploited, for example, by using transistors operating in the subthreshold regime to massively reduce energy consumption (Benjamin et al., 2014). However these physical models come at the cost of reduced precision in their operations. This is not necessarily a problem for integration of synaptic inputs due to locality and the vast number of arriving inputs, but can be a problem for distribution of spikes over larger distances. Analog neuron and synapse dynamics are hence combined with digital interconnects, that enable reliable transmission of action potentials (“aggregate analog – distribute digital”). Neuron and synapse models are fixed in these systems since the type of circuit that implements them cannot be easily modified, but they are typically parametrized to provide a certain degree of flexibility (Millner et al., 2010). The Spikey chip (Pfeil et al., 2013b), for example, implements leaky-integrate-and-fire neuron models with conductance-based synapses in VLSI circuits, available on a small printed-circuit board (PCB) that can be connected to a host computer via a regular USB connection. The chip houses 384 neurons that can be (almost) arbitrarily connected via plastic synapses implementing a particular STDP rule (Pfeil et al., 2012). Mixed-signal systems like Spikey can offer a speedup of up to $10^4\times$ biological realtime (Pfeil et al., 2013b), allowing investigations of neuronal network dynamics over biological times not just of minutes or hours, but even weeks or years. A follow-up system aims at wafer-scale integration of many neuromorphic chips to simulate about 200k adaptive-exponential neurons (Brette and Gerstner, 2005) each receiving up to 256 connections via plastic synapses on a single wafer (Schemmel et al., 2010a). In this system, multiple neuronal membranes can be coupled to implement few-compartment models and to increase the synaptic input count at the expense of a decreased effective number of individual neurons. One of the main challenges that analog implementations of neuron and synapse dynamics face are the high variability of analog components, on the one hand arising due to unavoidable fluctuations in the production process of the circuits (“fixed-pattern noise”), on the other hand for example from thermal noise giving rise to trial-to-trial variations (“temporal noise”). In order to compensate for fixed-pattern noise, neuromorphic systems often employ sophisticated calibration routines that rely on the configurability of the analog circuits (see, e.g., Schwartz, 2013), which however are limited in their accuracy. To successfully run network models on the hardware substrate the effect of both types of variability on network dynamics need to be thoroughly understood and new modeling paradigms, for example training network models directly in the presence of device variability, need to be developed (see, e.g., Schmitt et al., 2017). In chapter 6 we investigate in detail how fixed-pattern noise influences the dynamics of a recurrent network emulated on the Spikey chip. For other examples of modeling approaches in mixed-signal neuromorphic systems see Pfeil et al. (2013b), Schmuker, Pfeil, and Nawrot (2014), Petrovici et al. (2014a), Petrovici et al. (2017), and Schmitt et al. (2017).

Digital systems either rely on application-specific-integrated circuits (ASICs) that, similar to the analog approach, implement fixed dynamics with adjustable parameters (Cassidy et al., 2013) or rely on off-the-shelf microprocessors with a high degree of parallelism (Furber et al., 2014). Digital neuromorphic systems avoid issues of spatial or temporal noise by relying on fully digital signal processing, even allowing for a one-to-one correspondence between computer simulations of the hardware and the hardware system itself (Merolla et al., 2014). One of the main design targets of digital neuromorphic systems is low energy usage of simulations

running at biological realtime, for example to be deployed in robotic or mobile applications. Systems built around standard microprocessors provide the huge advantage over other approaches that, similar to the pure software approach, any type of neuron and synapse dynamics can be implemented, making them particularly well suited for neuroscientific research. The SpiNNaker system for example is built by integrating many energy-efficient general-purpose ARM cores on PCBs with a custom communication infrastructure using distributed routing tables and supporting globally-asynchronous-locally-synchronous (GALS) operation (Furber et al., 2014). Each chip is equipped with 128MB of memory that houses local neurons and synapses, 18 ARM cores, responsible for propagating neuron and synapse dynamics, and a multicast router receiving spikes from other chips, injecting spikes to the network or forwarding them according to the routing tables. Due to its dependence on traditional microprocessors, SpiNNaker can be seen as a supercomputer equipped with a special communication fabric optimized for small packets and extreme parallelism, thereby reducing the energy consumption and achieving simulations in biological realtime. The largest HPC system based on SpiNNaker boards totals 500 000 processor cores potentially simulating up to $460 \cdot 10^6$ neurons at biological realtime at the energy budget of 50 kW (Furber, 2016). For examples of modeling approaches in digital neuromorphic systems see Merolla et al. (2014), Stamatias et al. (2015b), Mundy et al. (2015), Mendat et al. (2016), and Van Albada et al. (2016).

An interesting research avenue into energy-efficient neuromorphic elements is given by memristors (Chua, 1971), an electronic component that changes its resistance based on voltages applied in the past and can thereby be used to emulate the integrative properties of neurons (Tuma et al., 2016) and the learning capabilities of synapses (Li et al., 2013), while requiring very little power and space. While initial progress was reported, further research is necessary to integrate these new materials with traditional CMOS to create scalable implementations. While for “black-box” machine learning approaches such systems hold much potential (Pantazi et al., 2016), for neuroscientific research, one particular difficulty with these new materials is the lack of fine-grained control over the different components of the system – the energy and space requirements for the control electronics would most likely mitigate the gain in efficiency by new materials.

This section only provided a glimpse of the field of neuromorphic engineering that currently is developing at an unprecedented pace. For example, this chapter does not discuss projects like the silicon retina (Delbruck, 1993; Posch et al., 2014) or silicon cochlea (Yang et al., 2016) that do not aim at emulating neuronal network dynamics, but rather seek inspiration from neurobiology to build powerful sensory devices that consume significantly less power than traditional approaches (Vanarse, Osseiran, and Rassau, 2016). Since further discussions are outside the scope of this thesis, we refer to the literature for in-depth reviews of current neuromorphic developments (Indiveri et al., 2011a; Furber, 2016; Nawrocki, Voyles, and Shaheen, 2016; Schuman et al., 2017).

Chapter 4

Scope and structure of this thesis

This thesis broadly considers two topics in the field of computational neuroscience: biophysical sources of stochasticity for approximate inference in neural circuits and scalable simulation technology for numerical investigation of neural network dynamics and function at cellular resolution. For each of these we consider a series of specific questions, which we shortly introduce in the following.

Can deterministic networks with a finite number of external noise sources approximate sampling in Boltzmann machines? As discussed in chapter 1, probabilistic inference has been proposed as a normative principle of brain function (Fiser et al., 2010; Pouget et al., 2013), with approximate inference as a computationally feasible approach in systems with restricted resources (Gershman, Horvitz, and Tenenbaum, 2015). Particular Markov-chain-Monte-Carlo-sampling algorithms can be interpreted as dynamics of recurrent neural networks of binary units leading to models amenable for implementation in neural substrates (Hinton, Sejnowski, and Ackley, 1984a; Buesing et al., 2011; Petrovici et al., 2016). Most of these models, however, rely on single unit dynamics that are either intrinsically stochastic (Buesing et al., 2011) or require private external sources of noise (Petrovici et al., 2016). These assumptions are hardly met in physical substrates, be they biological or neuromorphic implementations. In physical systems with space and bandwidth constraints units typically have to share noise sources, leading to shared-input correlations and violating the assumption of independence of noise across units. It is currently an open question in how far these shared-input correlations influence the functional performance of these models of approximate and, if they are detrimental, how one can mitigate their effect. One potential option to remove these shared-input correlations is by introducing a specific correlation structure among the activity of the noise sources. In recurrent neural networks, negative feedback naturally imposes a correlation structure in the network activity that actively suppresses correlations arising from shared inputs (Tetzlaff et al., 2012a). In chapter 5 we investigate in how far we can exploit this effect to supply sampling networks with uncorrelated background input even in the case of a finite number of noise sources.

How do network heterogeneities in neuromorphic hardware substrates influence correlations in recurrent networks? Our results regarding the first question have direct applications in the field of neuromorphic engineering: Due to space and bandwidth constraints in physical devices, generation and transmission of large amounts of random numbers is expensive. This causes severe challenges for neuromorphic implementations of approximate inference. Generating noise by a recurrent neural network emulated side-by-side with the sampling network on the neuromorphic chip, would mitigate these issues, but requires a specific correlation structure in the

recurrent network. It is unclear how heterogeneities due to fixed-pattern noise arising from unavoidable fluctuations in the production process in analog and mixed-signal neuromorphic systems (Pfeil et al., 2013b; Schwartz, 2013) influence correlations in networks emulated in these substrates. So far, studies have either focused on correlations in heterogeneous feedforward networks (Middleton et al., 2012; Ly, Middleton, and Doiron, 2012) or correlations in homogeneous feedback networks (Renart et al., 2010a; Tetzlaff et al., 2012a), but not in heterogeneous recurrent systems, and specifically not in concrete hardware implementations. In chapter 6 we investigate whether the effect of decorrelation through inhibitory feedback, responsible for the dynamic suppression of shared-input correlations, is also present in neural network emulated on the mixed-signal system Spikey (Pfeil et al., 2013b). This is a prerequisite for employing recurrent neural networks as noise sources in neuromorphic implementations.

Is fast inference via constraining network dynamics possible in non-Boltzmann networks? In Boltzmann machines (Hinton, Sejnowski, and Ackley, 1984b), one can constrain the network dynamics by manipulating parameters of the model such that the network generates samples directly from the exact conditionals (cf. chapter 2, and, e.g., Welling and Teh, 2003). As of yet it has not been systematically investigated in how far this approach depends on the specific sampling dynamics in Boltzmann machines or whether this computational trick applies to non-Boltzmann networks with different single unit dynamics or non-symmetric connectivity. Since the logistic activation function for single units in Boltzmann machines is difficult to achieve with natural noise sources, an important question is how far the ability to perform inference via constraints depends on the specific form of the activation function. In chapter 7 we hence consider the special case of Gaussian units, and investigate in how far networks of these units retain the ability to perform inference via constraints.

Biological neural network show trial-to-trial variability at various levels (Faisal, Selen, and Wolpert, 2008a) that potentially could serve as sources of stochasticity for approximate inference. In particular the variability in postsynaptic currents elicited by presynaptic spikes (Ribrault, Sekimoto, and Triller, 2011a; Branco and Staras, 2009) has recently received attention for its potential role for computations in neural networks (Rosenbaum, Rubin, and Doiron, 2012; Neftci et al., 2016; Muller and Indiveri, 2017). However, a systematic investigation of the representational power of networks of deterministic units with stochastic synaptic transmission and their potential for probabilistic computations is so far lacking. In chapter 7 we investigate a simple model of synaptic stochasticity and systematically reexamine claims from the literature regarding the suitability of synaptic stochasticity for approximate inference with a particular focus on differences arising from binary and bipolar units.

While network models typically considered for approximate inference in neural networks have symmetric connectivity (Hinton, Sejnowski, and Ackley, 1984a), biological neural networks are fundamentally asymmetric, if only due to the distinction of excitatory and inhibitory neurons (Abeles, 1991). Asymmetric networks have a significant advantage over symmetric models due to their non-equilibrium dynamics: the ability to represent spatio-temporal patterns. However, no closed form solution of the stationary distribution of network states is known, making them difficult to analyze systematically. As a first step in this direction we study representation of distributions and approximate inference in small asymmetric networks in chapter 7.

What are the limits of current neural simulation technology and is it scalable to post-petascale systems? To address neuroscientific questions such as those presented above, modern neuroscientific research is supported by well-tested, high-performance simulation tools like NEURON (Carnevale and Hines, 2006), NEST (Gewaltig and Diesmann, 2007) and BRIAN (Goodman and Brette, 2013). These allow researchers to focus on the neuroscientific aspects of their efforts, instead of software and infrastructure development. To be able to perform cellular-resolution simulations at the scale of human brains, modern simulators need to be able to scale to the biggest supercomputing systems available today. The current simulation technology can simulate networks of up to 10^9 neurons connected by 10^{14} synapses, corresponding to about 10% of human cortex (Kunkel et al., 2014). Full-scale simulations of the human cortex at a resolution of individual neurons and synapses are hence within reach of the next generation of supercomputers. To support these large-scale HPC facilities, simulation software needs to exhibit excellent scaling behavior over tens of thousands of MPI processes. In chapter 8 we investigate the limits of the current simulation technology as implemented in the NEST simulator and design a new connection infrastructure and communication scheme that exploits the sparsity of large-scale networks distributed over tens of thousands of compute nodes.

How can functional network models in neural simulators be provided with complex, non-stationary and reproducible stimuli? Functional network models for probabilistic inference typically consider tasks with stationary input, e.g., the MNIST data set (LeCun, Cortes, and Burges, 1998). This type of input is straightforward to provide to network models in neural simulators and can be compactly described in publications. However, our brains have to handle constantly changing input and for more realistic models one hence needs to consider complex, non-stationary stimuli. In general, this type of input is non-trivial to generate and researchers currently rely on manual ad-hoc implementations, making models difficult to compare and reproduce, ultimately leading to a fragmentation of research efforts. In the field of machine learning, standardized benchmarks and toolkits have significantly improved the comparability of models and allowed the community to quickly focus on promising approaches. Instead of reinventing the wheel, researchers in computational neuroscience should be able to leverage these tools for their own models. However, as machine learning toolkits and neural network simulators are generally designed for two different communities, they are non-trivial to interface. In chapter 9 we address this issue by developing a toolchain that allows researchers to interface the OpenAI Gym toolkit (Brockman et al., 2016) with MUSIC-compatible simulators (currently NEST and NEURON, Djurfeldt et al., 2010a), enabling training and evaluation of functional neural network models in neural simulators on standard environments from the OpenAI Gym.

Chapter 10 summarizes the results of this thesis, specifically addressing the questions posed above and in chapter 11 we conclude by discussing further research directions. Chapter A, chapter B, chapter C and chapter D in the appendix contain additional material for the individual chapters and detailed information on the numerical simulations performed in this thesis.

Part II

Results

Chapter 5

Stochastic neural computation without noise

This chapter is based on the following publication:

Jakob Jordan, Mihai A. Petrovici, Oliver Breitwieser, Johannes Schemmel, Karlheinz Meier, Markus Diesmann, and Tom Tetzlaff (2017b). “Stochastic neural computation without noise”. In: *ArXiv e-prints*. arXiv: 1710.04931 [q-bio.NC]

Author contributions:

Under supervision of Tom Tetzlaff and Markus Diesmann, the author performed work on all parts of the concepts of the study, the mathematical analysis, the simulation of binary networks, and the data analysis and interpretation. Spiking network simulations carried out by Oliver Breitwieser. All authors contributed to the writing of the manuscript.

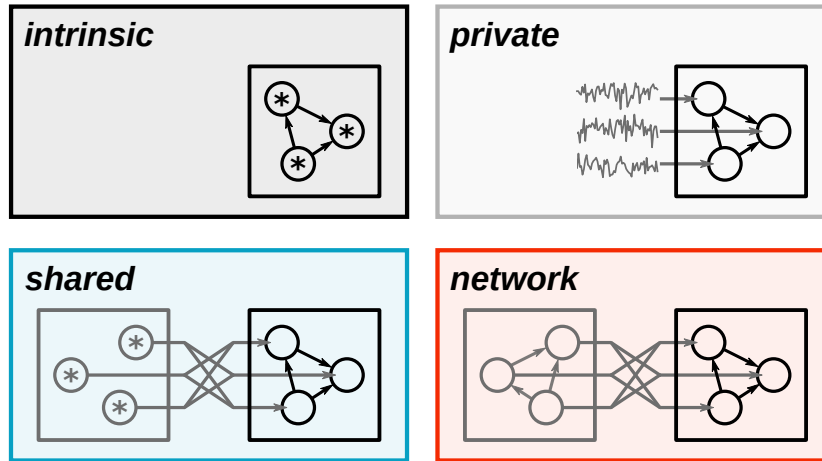


FIGURE 5.1: Sources of stochasticity (gray) for functional neural networks (black). Stars indicate intrinsically stochastic units. Open circles correspond to deterministic units. **Intrinsic:** Intrinsically stochastic units updating their binary states with a probability determined by their total synaptic input. **Private:** Deterministic units receiving private additive independent noise. **Shared:** Deterministic units receiving noise from a finite population of independent stochastic sources. **Network:** Deterministic units receiving quasi-random input generated by a finite recurrent network of deterministic units.

5.1 Introduction

The high in-vivo response variability of cortical neurons during sensory stimulation (Destexhe, Rudolph, and Paré, 2003) is by now a well-established fact and has often been interpreted in the context of ongoing probabilistic computation (Hoyer and Hyvärinen, 2003; Berkes et al., 2011; Orbán et al., 2016). While probabilistic inference as a principle for brain function has attracted more and more attention over the past decades, it has also been found that intrinsically stochastic neural networks can be highly performant substrates for machine learning (Hinton and Salakhutdinov, 2006b; Salakhutdinov and Hinton, 2009). Regardless of the purported source of stochasticity – noise in synaptic transmission (Branco and Staras, 2009), ion channel noise (White, Rubinstein, and Kay, 2000) or spiking background input (Holt et al., 1996) – these findings have led to the incorporation of noise mechanisms into computational neuroscience models (Burkitt, 2006a; Burkitt, 2006b), to give mechanistic accounts for stochastic inference in neural substrates.

The arguably most widespread implementation of noise in neural-network models happens at the level of individual neurons. Here, neurons are described as intrinsically stochastic units updating their (binary) states in a way that is uniquely determined by their synaptic input (figure 5.1, *intrinsic*) (Ackley, Hinton, and Sejnowski, 1985; Buesing et al., 2011; Habenschuss, Jonke, and Maass, 2013). Alternatively, deterministic neurons are equipped with an additive private independent noise (figure 5.1, *private*), often in the form of Gaussian white noise or random sequences of action potentials (spikes) modeled as Poisson point processes (Lundqvist et al., 2006b; Petrovici et al., 2016). In either case, each neuron essentially requires its own private pseudorandom number generator. The implicit assumption of independence of this background noise across units in the network is usually mentioned en passant and goes unchallenged.

However, feeding a large amount of uncorrelated noise into a physical realization of a system comes with costs. In particular, the supply of randomness is physically limited by the input bandwidth of the system, which is necessarily finite for systems that occupy a finite volume, be they biological or synthetic. This poses a fundamental problem for the biological plausibility of network models of probabilistic inference, as well as for their emulation on physical, neuromorphic devices (Indiveri et al., 2011b; Petrovici et al., 2014b). A straightforward solution is to limit the size of the available pool of noise sources (figure 5.1, shared). However, this invariably leads to sharing of noise sources among neurons (figure 5.2), which, in turn, violates the assumption of independence and potentially impairs the performance of the network.

The present work demonstrates that replacing the finite ensemble of independent noise sources by a recurrent neural network (figure 5.1, network) alleviates the problem of shared noise. In recurrent neural networks with dominant inhibition, shared-input correlations are dynamically suppressed by a specific correlation structure of network activity (Renart et al., 2010b; Tetzlaff et al., 2012b). We propose to exploit the effect to suppress shared-noise correlations in functional networks resulting from a finite number of noise sources (see figure 5.2). While being consistent with biology, the mechanism is particularly useful for the implementation of stochastic computing paradigms on large-scale neuromorphic platforms (Schemmel et al., 2010b; Furber et al., 2013).

Neural network implementations derived from Boltzmann machines (Ackley, Hinton, and Sejnowski, 1985) are representative examples of stochastic network models. Such networks are widely used in machine learning (Hinton and Salakhutdinov, 2006b; Salakhutdinov and Hinton, 2009), but also in theoretical neuroscience as models of brain dynamics and function (Ginzburg and Sompolinsky, 1994; Buesing et al., 2011; Petrovici et al., 2016). For the purpose of the present study, the advantage of models of this class lies in our ability to quantify their functional performance when subject to limitations in the quality of the noise sources.

5.2 Methods

5.2.1 Binary network simulation

Sampling networks consist of M binary units that switch from the inactive (0) to the active (1) state with a probability $F_i(h_i) := p(s_i = 1|h_i)$, also referred to as the “activation function”. The input field h_i of a unit depends on the state of the presynaptic units and is given by:

$$h_i(\mathbf{s}) = \sum_j w_{ij}s_j + b_i. \quad (5.1)$$

Here w_{ij} denotes the weight of the connection from unit j to unit i and b_i denotes the bias of unit i . We perform an event-driven update, drawing subsequent inter-update intervals $\tau_i \sim \text{Exp}(\lambda)$ for each unit from an exponential distribution with rate $\lambda := 1/\tau$ with an average update interval τ . Starting from $t = 0$, we update the neuron with the smallest update time t_i , choose a new update time for this unit $t_i + \tau_i$ and repeat this procedure until any t_i is larger than the maximal simulation duration $T_{\max}$.

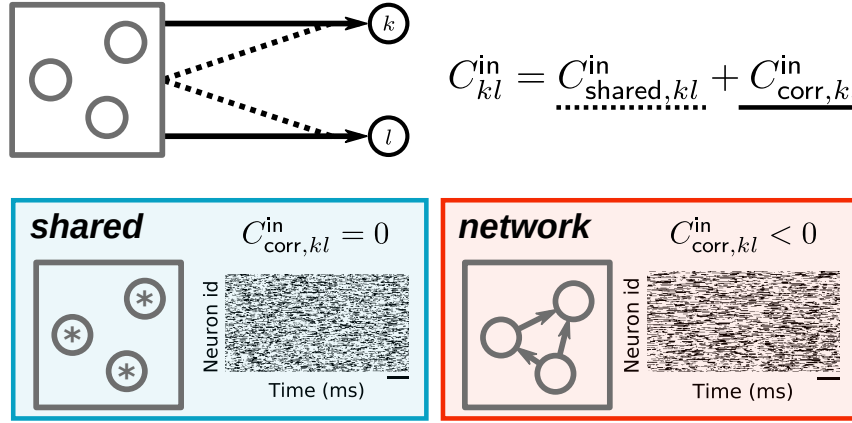


FIGURE 5.2: Origin of shared-input correlations and their suppression by correlated presynaptic activity. **Top:** A pair of neurons k and l (black) receiving input from a finite population of noise sources (gray). The input correlation C_{kl}^{in} can be decomposed into a contribution $C_{shared,kl}^{in}$ resulting from shared noise sources (dotted lines) and a contribution $C_{corr,kl}^{in}$ (solid lines) due to potential correlations between sources. If inputs obey Dale’s law (they are either excitatory or inhibitory), shared-input correlations are always positive ($C_{shared,kl}^{in} > 0$). **Bottom left:** In the shared scenario, sources are by definition uncorrelated ($C_{corr,kl}^{in} = 0$) and cannot compensate for shared-input correlations. **Bottom right:** In inhibition-dominated neural networks (network case), correlations between units arrange such that $C_{corr,kl}^{in}$ is negative, thus being able to compensate for shared-input correlations such that $C_{kl}^{in} \approx 0$ (Renart et al., 2010b; Tetzlaff et al., 2012b). Raster displays in the bottom row illustrate activity of binary noise sources in the shared (left) and the network scenario (right). Scale bars correspond to 200 ms.

5.2.2 Random sampling networks

Weights are randomly drawn from a beta distribution $\text{Beta}(a, b)$ and shifted to have mean μ_{BM} . Weights are symmetric ($w_{ij} = w_{ji}$) and self connections are absent ($w_{ii} = 0$). To control the average activity in the network, the bias for each unit is chosen such that on average, it cancels the input from the other neurons in the network for a desired average activity $\langle s \rangle$: $b_i = M\mu \langle s \rangle$ (Helias, Tetzlaff, and Diesmann, 2014b). Whenever a unit is updated, the state of (a subset) of all units in the sampling network is recorded. To remove the influence of initial transients, i.e., the burn-in time of the Markov chain, samples during the initial interval of each simulation (T_{warmup}) are excluded from the analysis. From the remaining samples we compute the empirical distribution p of network states. The following sections introduce the activation function for the units for different ways of introducing noise to the system.

5.2.3 Intrinsic noise

Intrinsically stochastic units switch to the active state with probability

$$F_i(h_i) = \frac{1}{1 + e^{-\beta h_i}}, \quad (5.2)$$

where β determines the slope of the logistic function and is also referred to as the “inverse temperature”. For small β , changes in the input field have little influence of the update probability, while for large beta a unit is very sensitive to changes in h_i and in the limit $\beta \rightarrow \infty$ the activation function becomes a Heaviside step function.

Symmetric networks with these single-unit dynamics and the update schedule described in section 5.2.1 are identical to Boltzmann machines, leading to a stationary distribution of network states of Boltzmann form:

$$p(\mathbf{s}) \sim \exp \left(\frac{\beta}{2} \sum_{i,j} w_{ij} s_i s_j + \beta \sum_i b_i s_i \right). \quad (5.3)$$

Instead of directly prescribing a stochastic update rule like equation (5.2), we can view these units as deterministic units with a Heaviside activation function and additive noise on the input field:

$$F_i(h_i) = \Theta(h_i + \xi_i),$$

with $\xi_i \sim 1/2(1 - \tanh^2(\xi_i))$ (Coolen, 2001) and Θ denoting the Heaviside step function

$$\Theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases} \quad (5.4)$$

Averaging over the noise ξ_i yields the probabilistic update rule (equation (5.2)). However, on biophysical grounds it is difficult to argue for this particular distribution of the noise.

5.2.4 Private noise

We consider a deterministic model in which we assume a more natural distribution for the additive noise, namely Gaussian form ($\xi_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$), for example arising from a large number of independent background inputs (Hinton, Sejnowski, and Ackley, 1984b). In this case, the noise averaged activity for fixed h_i is given by:

$$\begin{aligned} F_i(h_i) &= \int_{-\infty}^{\infty} d\xi_i \Theta(h_i + \xi_i) p(\xi_i) \\ &= \int_{-h_i}^{\infty} d\xi_i \mathcal{N}(\mu_i, \sigma_i^2) \\ &= \frac{1}{2} \operatorname{erfc} \left(-\frac{h_i + \mu_i}{\sqrt{2}\sigma_i} \right). \end{aligned} \quad (5.5)$$

Similar to the intrinsically stochastic units (section 5.2.3), the update rule for deterministic units with Gaussian noise is effectively probabilistic. Both functions, share some general properties (bounded, monotonic):

$$\begin{aligned} \lim_{h_i \rightarrow -\infty} F_i(h_i) &= 0, \\ \lim_{h_i \rightarrow \infty} F_i(h_i) &= 1, \\ \partial_{h_i} F_i(h_i) &> 0 \quad \forall h_i, \end{aligned}$$

and one can hence hope to approximate the dynamics in Boltzmann machines with a network of deterministic units with Gaussian noise by a systematic matching of parameters.

One approach is to choose parameters for the Gaussian noise such that the difference between the two activation functions is minimized. To simplify notation

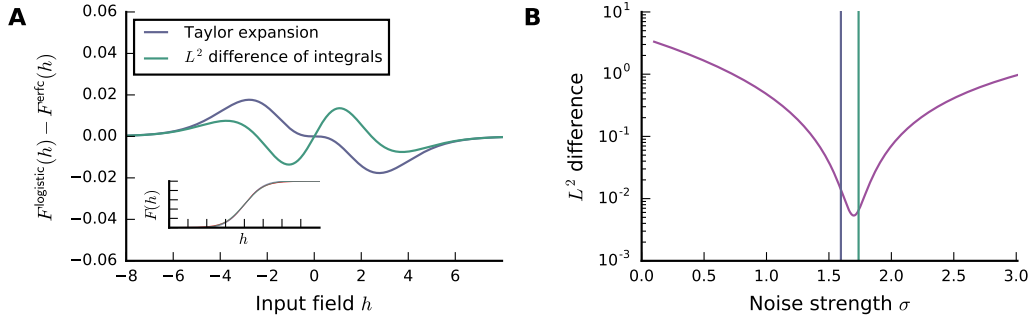


FIGURE 5.3: Fit of error function to logistic function via Taylor expansion (purple) and L^2 difference of integrals (green). **A** Difference of logistic activation function and error function with adjusted σ via equation (5.6) (purple) and via equation (5.9) (green). Inset: activation functions. **B** L^2 difference of activation functions (equation (5.7)) as a function of the strength of the Gaussian noise σ . Vertical bars indicate σ obtained via the respective method.

we drop the index i in the following calculations. Since both activation functions are symmetric around zero, we require that their value at $h = 0$ is identical, fixing one parameter of the noise distribution ($\mu = 0$). To find an expression for the noise strength σ , the simplest method equates the coefficients of a Taylor expansion up to linear order of both activation functions around zero. For the logistic activation function (equation (5.2)) this yields:

$$F(h) = 0.5 + 0.25\beta h + \mathcal{O}(h^2),$$

while for the units with Gaussian noise (equation (5.5)) we obtain

$$F(h) = 0.5 + \frac{1}{\sqrt{2\pi}\sigma} h + \mathcal{O}(h^2).$$

Equating the coefficients of h gives an expression for the noise strength σ as a function of the inverse temperature β :

$$\sigma(\beta) = \frac{2\sqrt{2}}{\sqrt{\pi}\beta}. \quad (5.6)$$

While this approach is conceptually simple, the Taylor expansion around zero leads to large deviations between the activation functions for input fields different from zero (figure 5.3).

Another option taking into account all possible values of h is to minimize the L^2 difference of the two activation functions:

$$\sigma = \arg \min_{\sigma'} \int dh (l(h) - g(h, \sigma'))^2, \quad (5.7)$$

where l denotes the logistic and g the activation function for Gaussian noise. Since it is not possible to analytically evaluate the resulting integral, we opt for a slightly simpler approach: minimizing the L^2 difference of integrals of the activation function from $-\infty$ to 0:

$$\sigma = \arg \min_{\sigma'} \left(L(h) \Big|_{-\infty}^0 - G(h, \sigma') \Big|_{-\infty}^0 \right)^2,$$

with capital letters denoting antiderivatives. To find the minimal σ , we take the derivative of the right hand side with respect to σ' and equate it with zero:

$$-2 \left(F(h) \Big|_{-\infty}^0 - G(h) \Big|_{-\infty}^0 \right) \partial_{\sigma} G(h) \Big|_{-\infty}^0 = 0.$$

From this we observe that

$$\left(F(h) \Big|_{-\infty}^0 - G(h) \Big|_{-\infty}^0 \right) = 0, \quad (5.8)$$

is a sufficient condition to satisfy this equation. We compute the integral of both activation functions. For the logistic activation function (equation (5.2)) we obtain:

$$\begin{aligned} \int dh F(h) &= \int dh \frac{1}{1 + e^{-\beta h}} \\ &= h + \frac{\log(1 + e^{-\beta h})}{\beta}, \end{aligned}$$

with the definite integral

$$\int_{-\infty}^0 dh F(h) = \frac{\log 2}{\beta},$$

since the two diverging terms for $h \rightarrow -\infty$ cancel. For the activation function with Gaussian noise (equation (5.5)) we get:

$$\begin{aligned} \int dh F(h) &= \int dh \frac{1}{2} \operatorname{erfc} \left(\frac{-h}{\sqrt{2}\sigma} \right) \\ &= \frac{\sigma}{\sqrt{2\pi}} e^{-\frac{h^2}{2\sigma^2}} + 0.5 h \operatorname{erfc} \left(\frac{-h}{\sqrt{2}\sigma} \right), \end{aligned}$$

and computing the definite integral leads to:

$$\int_{-\infty}^0 dh F(h) = \frac{\sigma}{\sqrt{2\pi}},$$

since the second term vanishes for $h \rightarrow -\infty$ as the complementary error function decreases faster than $|h|^{-1}$. From equation (5.8) we hence find σ as a function of β :

$$\sigma(\beta) = \frac{\log 2 \sqrt{2\pi}}{\beta}. \quad (5.9)$$

Even though this value is not minimizing the L^2 difference, it provides a better fit than that obtained by simply Taylor expanding around zero, since in this case we are also taking into account the mismatch for larger absolute values of h (figure 5.3). We will hence use equation (5.9) to translate between the inverse temperature β of the logistic activation function and the strength σ of the Gaussian noise.

5.2.5 Shared noise

In the previous section we have assumed that each deterministic unit in the sampling network receives private, uncorrelated Gaussian noise. Now we instead consider a

second population $\mathcal{B}$ of $N = |\mathcal{B}|$ mutually unconnected, intrinsically stochastic units with logistic activation functions (cf. section 5.2.3) that provide additional input to units in the sampling network. In the following we will denote the population/set of units in the sampling network by $\mathcal{S}$ and refer to the second population as the background population or noise population. The input field for a unit i in the sampling network $\mathcal{S}$ hence contains an additional term arising from projections from the background population (cf. equation (5.1)):

$$h'_i = \underbrace{\sum_{j \in \mathcal{S}} w_{ij} s_j + b_i}_{h_i} + \underbrace{\sum_{k \in \mathcal{B}} m_{ik} z_k}_{\text{background input}} . \quad (5.10)$$

Here z_k denotes the state of the k th unit in the background population $\mathcal{B}$ and m_{ij} the weight from unit j in the background population to unit i in the sampling network. Given the total input field h'_i , the neurons in the sampling network change their state deterministically, according to

$$F_i(h'_i) = \Theta(h'_i) . \quad (5.11)$$

Since the units in the background population are mutually unconnected, their average activity $\langle z_i \rangle$ can be arbitrarily set by adjusting their bias: $b_k = F^{-1}(\langle z_k \rangle)$, where F^{-1} denotes the inverse of the logistic activation function:

$$F^{-1}(\langle z \rangle) = \frac{1}{\beta} \log \frac{1}{\frac{1}{\langle z \rangle} - 1} .$$

Ignoring the actual state of the background population, we can employ the central limit theorem and approximate the background input in the input field h'_i by a normal distribution with mean and variance given by

$$\mu_i = \sum_{k \in \mathcal{B}} m_{ik} \langle z_k \rangle , \quad (5.12)$$

$$\sigma_i^2 = \sum_{k \in \mathcal{B}} m_{ik}^2 \langle z_k \rangle (1 - \langle z_k \rangle) . \quad (5.13)$$

The total input field can then be written as $h'_i = h_i + \xi_i$ with $\xi_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$, as in the case of private uncorrelated Gaussian noise. However, note that correlations in input fields h'_i and h'_j in the sampling network arise due to units in the background population projecting to multiple units in the sampling network ($\langle (\xi_i - \mu_i)(\xi_j - \mu_j) \rangle$ does not necessarily vanish for all $i, j \in \mathcal{S}$).

For the connections from the background population we use fixed weights and impose Dale's law, i.e., units are either excitatory $m_{ij} = w > 0 \ \forall i$ or inhibitory $m_{ij} = -gw < 0 \ \forall i$, with a ratio of excitatory units of $\gamma = |\mathcal{B}_E|/|\mathcal{B}|$. Here $w \in \mathbb{R}^+$ denotes the excitatory synaptic weight and $g \in \mathbb{R}^+$ a scaling factor for the inhibitory weights. Each unit $i \in \mathcal{S}$ in the sampling network receives exactly $K = \epsilon N$ inputs from units in the background population. $\epsilon = K/N \in [0, 1]$ is referred to as the connectivity. We do not allow multiple connections between a unit in the sampling network and unit in the background population. Assuming all units in the background population have identical average activity $\langle z \rangle$, all units in the sampling network receive statistically identical input and, in addition, the equations for the

mean and variance simplify to

$$\mu = Kw(\gamma - (1 - \gamma)g)\langle z \rangle, \quad (5.14)$$

$$\sigma^2 = Kw^2(\gamma + (1 - \gamma)g^2)\langle z \rangle(1 - \langle z \rangle). \quad (5.15)$$

We can hence employ the same procedure as in the previous section to relate the strength of the background input to the inverse temperature of a Boltzmann machine.

5.2.6 Network noise

We now consider a background population of deterministic units projecting to the sampling network. The background population has sparse, random, recurrent connectivity with a fixed indegree. Connections in the background population are realized with the same indegrees K , weights w and $-gw$ and ratio of excitatory inputs γ as the connections to the sampling network (cf. section 5.2.5). The connection matrix of the background population is hence generally asymmetric. As before, we can approximate the additional contribution to the input fields of neurons in the sampling network with a normal distribution, with parameters

$$\mu_i = \sum_{k \in \mathcal{B}} m_{ik} \langle z_k \rangle, \quad (5.16)$$

$$\sigma_i^2 = \sum_{k \in \mathcal{B}} m_{ik}^2 \langle z_k \rangle (1 - \langle z_k \rangle) + \sum_{k \neq l} m_{ik} m_{il} c_{kl}, \quad (5.17)$$

where the additional term in the input variances arises from correlations $c_{kl} := \langle (z_k - \langle z_k \rangle)(z_l - \langle z_l \rangle) \rangle$ between units in the background population. As in the sampling network we choose the bias to cancel the expected average input from other units in the network for a desired mean activity $\langle z_k \rangle$. However since the second population exhibits rich dynamics due to its recurrent connectivity the actual average activity will deviate from this value, in particular due to an influence of correlations on the mean activity. We employ an iterative meanfield-theory approach that allows us to compute average activities and average correlations approximately from the statistics of the connectivity. We now shortly summarize this approach following Helias, Tetzlaff, and Diesmann (2014b). Note that in the literature a threshold variable θ_i is often used instead the bias b_i , which differs in the sign: $b_i = -\theta_i$.

For a network of binary units, the joint distribution of network states $p(\mathbf{s})$ contains all information necessary to statistically describe the network activity, in particular mean activities and correlations. It can be obtained by solving the Master equation of the system, which determines how the probability masses of network states evolve over time in terms of transition probabilities between different states (Kelly, 2011)

$$\partial_t p(\mathbf{s}_i) = \sum_j p(\mathbf{s}_i | \mathbf{s}_j) p(\mathbf{s}_j) - p(\mathbf{s}_j | \mathbf{s}_i) p(\mathbf{s}_i). \quad (5.18)$$

The first term describes probability mass moving into state i from other states j and the second term probability mass moving from state i to other states j . Since in general, and in particular in large networks, equation (5.18) is too difficult to solve directly, we focus on obtaining equations for first two momenta of $p(\mathbf{s})$. Starting from the master equation one can derive the following self-consistency equations for the mean activity of units in a homogeneous network by assuming fluctuations

around their mean input to be statistically independent (Helias, Tetzlaff, and Diesmann, 2014b):

$$\partial_t \langle s_i \rangle + \langle s_i \rangle = \frac{1}{2} \operatorname{erfc} \left(-\frac{\mu_i + b_i}{\sqrt{2}\sigma_i} \right)$$

where the μ_i and σ_i are given by equation (5.16) and equation (5.17), respectively. To obtain the average activity in the stationary state, i.e., for $\partial_t \langle s_i \rangle = 0$, this equation needs to be solved self-consistently since the activity of unit i can influence its input statistics (μ_i, σ_i) through the recurrent connections. By assuming homogeneous excitatory and inhibitory populations, the N dimensional problem reduces to a two-dimensional one (Helias, Tetzlaff, and Diesmann, 2014b):

$$\langle s_\alpha \rangle = \frac{1}{2} \operatorname{erfc} \left(-\frac{\mu_\alpha + b_\alpha}{\sqrt{2}\sigma_\alpha} \right) \quad (5.19)$$

with $\alpha \in \{\mathcal{E}, \mathcal{I}\}$. The population-averaged equations for the mean and variance of the input hence are (Helias, Tetzlaff, and Diesmann, 2014b):

$$\mu_\alpha = \sum_{\beta} K_{\alpha\beta} w_{\alpha\beta} s_\beta, \quad (5.20)$$

$$\sigma_\alpha^2 = \sum_{\beta} K_{\alpha\beta} w_{\alpha\beta}^2 a_\beta + \sum_{\beta, \gamma} (Kw)_{\alpha\beta} (Kw)_{\alpha\gamma} c_{\beta\gamma}, \quad (5.21)$$

with $K_{EE} = K_{IE} = \gamma N$, $K_{EI} = K_{II} = (1 - \gamma)N$ and $w_{EE} = w_{IE} = w$, $w_{EI} = w_{II} = -gw$. To derive a self-consistency equation for pairwise correlations from the master equation one linearize the threshold activation function by considering a Gaussian distribution of the input field caused by recurrent inputs. This leads to the following set of linear equations for the population-averaged covariances (Helias, Tetzlaff, and Diesmann, 2014b):

$$2c_{\alpha\beta} = \sum_{\gamma} (\tilde{w}_{\alpha\gamma} c_{\gamma\beta} + \tilde{w}_{\beta\gamma} c_{\gamma\alpha}) + \tilde{w}_{\alpha\beta} \frac{a_\beta}{N_\beta} + \tilde{w}_{\beta\alpha} \frac{a_\alpha}{N_\alpha}, \quad (5.22)$$

with

$$c_{\beta\gamma} = \begin{cases} \frac{1}{N_\beta(N_\beta-1)} \sum_{i,j \in \beta, i \neq j} c_{ij} & \text{if } \beta = \gamma \\ \frac{1}{N_\beta N_\gamma} \sum_{i \in \beta, j \in \gamma} c_{ij} & \text{else} \end{cases}$$

The effective population-averaged weights $\tilde{w}_{\alpha\beta}$ are defined as:

$$\tilde{w}_{\alpha\beta} := S(\mu_\alpha, \sigma_\alpha) K_{\alpha\beta} w_{\alpha\beta},$$

with the susceptibility given by $S(\mu_\alpha, \sigma_\alpha) := \frac{1}{\sqrt{2\pi}\sigma_\alpha} \exp \left(-\frac{(\mu_\alpha + b_\alpha)^2}{2\sigma_\alpha^2} \right)$ (Helias, Tetzlaff, and Diesmann, 2014b). Since the average activity and covariances are mutually dependent, we employ an iterative numerical scheme in which we first determine the stationary activity under the assumption of zero correlations according to equation (5.19). Using this result we compute the population-average covariances from equation (5.22) which in turn can be used to improve the estimate for the stationary activity since they influence input statistics according to equation (5.21). We repeat this procedure until the values for population-averaged activities and covariances in two subsequent iterations do not differ significantly any more. The mean

activity and correlations in the recurrent background population obtained via this procedure, allows us to compute the input statistics in the sampling network and hence relate the inverse temperature to the mean and variance of the input as in section 5.2.4.

Certain assumptions enter this analytical description of network statistics, which might not be fulfilled in general. The description becomes much more complicated for spiking neuron models with non-linear subthreshold dynamics like conductance-based neurons in neuromorphic systems (Petrovici et al., 2016). In this case, one can resort to empirically measuring the input statistics for a single isolated neuron given a certain arrangement of background sources (cf. section 5.2.9). An advantage of this methods is that it is easy and straight forward to implement and will work for any configuration of background populations and sampling networks, allowing for arbitrary neuron models and parameters. However, to estimate the statistics of the input accurately, one needs to collect statistics over a significant amount of time.

5.2.7 Calibration (binary networks)

The methods discussed above allow us to compute effective inverse temperature β_{eff} from the statistics of different background inputs, either additive Gaussian noise, a population of intrinsically stochastic units or a recurrent network of deterministic units. To approximate Boltzmann distributions via samples generated by networks with noise implemented via these alternative methods, we match their (effective) inverse temperatures. A straightforward option is to adjust the noise parameter according to the desired input statistics. While this is possible in the case of additive Gaussian noise for which we can freely adjust μ_i and σ_i , it is difficult to achieve in practice for the other methods. We can achieve the same effect by rescaling the weights and biases in the sampling network. The inverse temperature β appears as a multiplicative factor in front of weights and biases in the stationary distribution of network states (equation (5.3)). Scaling β is hence equivalent to scaling all weights and biases by the inverse factor (see also Grytskyy et al. (2013b), Dahmen, Bos, and Helias (2016), and Petrovici et al. (2016)). An infinite amount of Boltzmann machines hence exists, all differing in weights ($w \rightarrow \alpha w, \alpha \in \mathbb{R}^+$), biases ($b \rightarrow \alpha b$) and inverse temperatures ($\beta \rightarrow \beta/\alpha$), producing statistically identical samples. Given a mean background input μ_i and an effective inverse temperature $\beta_{\text{eff}}(\sigma_i)$ (cf. equation (5.9)) arising from a particular realization of noise sources, we can emulate a Boltzmann machine at inverse temperature β by rescaling all weights and biases in the sampling network according to

$$b_i \rightarrow \beta/\beta_{\text{eff}} b_i - \mu_i, \quad (5.23)$$

$$w_{ij} \rightarrow \beta/\beta_{\text{eff}} w_{ij}. \quad (5.24)$$

This method hence only requires us to adapt weights and biases globally in the sampling network according to the statistics arising from an arbitrary realization of background input.

5.2.8 Handwritten-digit generation

In the generative task, we measure how well a sampling network with various realizations of background noise can approximate a trained data distribution, in contrast to the random distributions considered in the other simulations. We use contrastive

divergence (CD-1, (Hinton, 2002)) to train a Boltzmann machine on a specific dataset. We consider a dataset consisting of a subset of MNIST digits (LeCun, 1998), down-scaled to 12x12 pixels and with grayscale values converted to black and white. We select one representative from each class ($0 \dots 9$) and extend the 144 array determining the pixel values with 10 entries for a one-hot encoding of the corresponding class, e.g., for the pattern zero, the last ten entries contain a 1 at the first place and zeros otherwise. These ten 154 dimensional patterns form the prototype dataset. A (noisy) training sample is generated by flipping every pixel from the first 144 entries of a prototype pattern with probability p^{flip} . After training, the network should represent a particular distribution q^* over classes. Training directly on a samples generated according to the class distribution q^* will, in general, lead to a different stationary distribution of one-hot readout states p generated by the network, since some patterns are more salient than others. For example, by training on equal amounts of patterns of zeros and ones, the network will typically generate more zero states. To nevertheless represent q^* with the network, we iteratively train the Boltzmann machine choosing images and labels from a distribution q that is adjusted between training sessions (algorithm 1). Over many repetitions this procedure will lead to a stationary distribution of classes p that closely approximates q^* .

Algorithm 1 Training of a fully visible Boltzmann machine via CD-1 to represent a particular distribution q^* over label units with one-hot encoding.

```

BM  $\leftarrow$  Boltzmann machine with random  $w, b$ 
 $q \leftarrow q^*$ 
for  $n_{\text{it}}$  iterations do
    collect  $n_{\text{sm}}$  samples of label-unit states from BM
     $p \leftarrow$  marginal distribution over one-hot label states
     $q \leftarrow [(1 - \alpha)q + \alpha q^* + \beta(q^* - p)]_+$  and normalize
     $d \leftarrow$  generate training samples with class frequencies  $q$ 
    train BM with CD-1 on dataset  $d$  over  $n_{\text{ep}}$  episodes
end for

```

After training a Boltzmann machine using this approach, we obtain a set of parameters, w and b , that can be translated to parameters for sampling networks by appropriate rescaling as discussed above. We collect samples by running the network in the absence of any input and recording the states of all label units.

5.2.9 Calibration (spiking networks)

Similar as for binary units, we need to match the parameters for spiking sampling networks to their respective counterparts in Boltzmann machines. Similar to Petrovici et al. (2016) we use high-rate excitatory and inhibitory inputs to turn the deterministic behavior of a leaky-integrate-and-fire neuron into an effectively stochastic response. However, in contrast to the original publication, we consider current based synapses for simplicity. Since the calibration is performed on single cell level, we use the identical calibration scheme for the private, shared and network case. For a given configuration of noise sources, we first simulate the noise network with the specified parameters and measure its average firing rate. The corresponding independent Poisson sources are set to fire with the same rate to ensure comparability between the two approaches. The calibrations are then performed by varying the resting potential and recording the average activity of a single cell that is supplied with input from either a noise network or Poisson sources. The private case

is calibrated separately in a similar manner. By fitting the logistic function to the activation obtained by this procedure, we obtain two parameters, a shift and a scaling parameter, which are used to translate the synaptic weights from binary units to spiking neurons. See Petrovici et al. (2016) for details of the translation between the two domains.

5.3 Results

5.3.1 Networks with private Gaussian noise approximate Boltzmann machines

Boltzmann machines (BMs, see Ackley, Hinton, and Sejnowski (1985)) are symmetrically connected networks of intrinsically stochastic binary units. With an appropriate update schedule and parametrization, the network dynamics effectively implement Gibbs sampling from arbitrary Boltzmann distributions (Geman and Geman, 1984b). A given network realization leads to a particular frequency distribution of network states. Efficient training methods (Hinton, 2002; Hinton and Salakhutdinov, 2006b) can fit this distribution to a given data distribution by modifying network parameters. In the following we investigate to what extent the functional performance of BM-like stochastic networks is altered if the intrinsic stochasticity assumed in BMs is replaced by private, shared or network-generated additive noise (figure 5.1). If not otherwise indicated, we consider BMs with random weights $w_{ij} \sim \text{Beta}(2, 2)$ (see SI) since these are easy to parametrize and do not require an extra training step. Due to the specific noise-generation processes, the neural network implementations deviate from the mathematical definition of a BM. We therefore refer to these implementations as “sampling networks”.

In BMs, the intrinsically stochastic units $i \in \{1, \dots, M\}$ are activated according to a logistic function $F_i(h_i) = (1 + e^{-\beta h_i})^{-1}$ of their input field $h_i = \sum_{j=1}^M w_{ij}s_j + b_i$ with inverse temperature β , presynaptic activity $s_j \in \{0, 1\}$, and bias b_i (see SI). In contrast to this intrinsic stochasticity, a more natural model considers additive Gaussian noise $\xi_i \sim \mathcal{N}(\mu, \sigma^2)$ on the input to deterministic neurons with Heaviside activation function $F_i(h_i) = \Theta(h_i + \xi_i)$ (Bryant and Segundo, 1976; Hinton, Sejnowski, and Ackley, 1984b; Mainen and Sejnowski, 1995a); (figure 5.1). Essentially normally distributed input naturally emerges in units receiving a large number of inputs from uncorrelated sources. Deterministic units receiving private Gaussian noise resemble units with a probabilistic update rule. However, in contrast to units in BMs, their effective activation function is a shifted error function $F_i(h_i) = \text{erfc}\left(h_i + \mu_i / \sqrt{2\sigma^2}\right) / 2$. We minimize the mismatch between the two activation functions by relating the standard deviation σ of the Gaussian noise to the inverse temperature β (see SI). For a given noise strength, this defines an effective inverse temperature β_{eff} . To emulate a BM at inverse temperature β , we rescale all weights and biases: $b_i \rightarrow \beta / \beta_{\text{eff}} b_i - \mu_i$, $w_{ij} \rightarrow \beta / \beta_{\text{eff}} w_{ij}$. We quantify the sampling error by the Kullback-Leibler divergence $D_{\text{KL}}(p, p^*)$ between the empirical state distribution p of the sampling network and the state distribution p^* generated by a BM over a subset of m units.

For matched parameters, networks of deterministic units with additive Gaussian noise closely approximate BMs (figure 5.4, gray vs. black). The sampling error decreases as a function of the sampling duration T , and saturates at a small but finite value (figure 5.4a, gray) due to remaining differences in the activation functions. The residual differences between the stationary distributions (figure 5.4b, black vs. gray

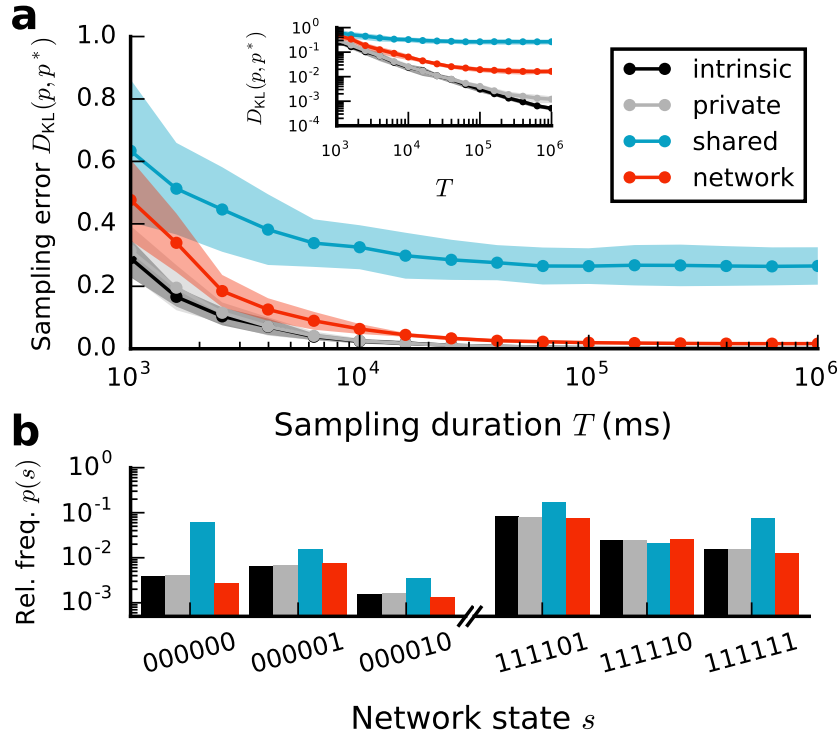


FIGURE 5.4: (a) Sampling error $D_{KL}(p, p^*)$ as a function of the sampling duration T for different sources of stochasticity (legend, cf. figure 5.1). Error bands indicate mean $\pm$ SEM over 5 random network realizations. Inset: Same data as main panel with log-log scaling. (b) Relative frequencies (log scale) of six exemplary states s for $T = 10^6$ ms. Parameters: $\beta = 1$, $M = 100$, $K = 200$, $N = 222$ (see SI).

bars) are significantly smaller than the differences in relative probabilities of different network states.

The assumption of private Gaussian noise generated by pseudo-random number generators is hard to reconcile with biology and difficult to achieve in current neuromorphic implementations. In the following, the Gaussian noise is replaced by input from binary and, subsequently, spiking units. As a consequence, the noise of the sampling units exhibits jumps with finite amplitudes determined by the weights of the incoming connections. Only if the number of input events per sampling unit is large and the weights are small, the collective signal resembles Gaussian noise. The sampling error resulting from private Gaussian noise therefore constitutes a lower bound on the error achievable by sampling networks supplied with noise from a finite pool of binary or spiking sources.

5.3.2 Shared-noise correlations impair sampling performance

Both in vivo and in silico, neurons in functional networks have to share noise sources to achieve a high input count per sampling unit in spite of space and bandwidth constraints. By replacing private noise with a large number of inputs from a finite pool of independent noise sources, we investigate how strongly the resulting shared-noise correlations distort the sampled distribution of network states. The noise sources are stochastic binary units with an adjustable average activity $\langle z \rangle$. To achieve a high input event count, each sampling unit is randomly assigned K inputs. For each unit, these are randomly chosen from a common pool of N sources.

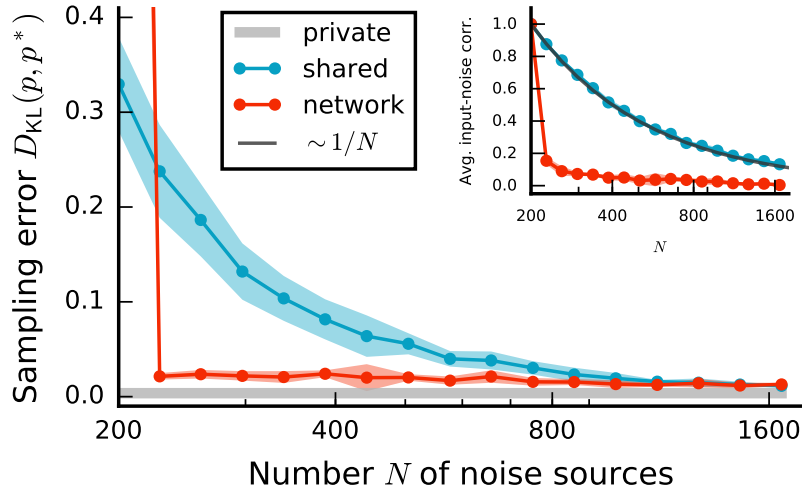


FIGURE 5.5: Sampling error $D_{KL}(p, p^*)$ as a function of the number N of noise sources for shared and network case. Error bands indicate mean $\pm$ SEM over 5 random network realizations. Inset: Dependence of average input correlation coefficient ρ of mutually unconnected sampling units on N . Black curve represents $\sim 1/N$ fit. Sampling duration $T = 10^5$ ms. Remaining parameters as in figure 5.4.

On average, a pair of neurons in the sampling network hence shares K^2/N identical noise sources. The ensemble of noise sources is comprised of γN excitatory and $(1 - \gamma)N$ inhibitory units, projecting to their targets with weights w and $-gw$, respectively. The input field for a single unit in the sampling network is then given by $h'_i = \sum_{j=1}^M w_{ij}s_j + b_i + \sum_{k=1}^N m_{ik}z_k$, where m_{ik} represents the strength of the connection from the k th noise source to the i th sampling unit. For homogeneous connectivity, the second term in h'_i can be approximated by a normal distribution with mean $\mu = Kw(\gamma - (1 - \gamma)g) \langle z \rangle$ and variance $\sigma^2 = Kw^2(\gamma + (1 - \gamma)g^2) \langle z \rangle (1 - \langle z \rangle)$ (see SI).

If $K \approx N$, shared-input correlations are large and the sampling error is substantial, even for long sampling duration (figure 5.4, blue curve and bars). Increasing N while keeping K fixed leads to a gradual decrease of shared-input correlations ($\sim 1/N$) and therefore to a reduction of the sampling error (figure 5.5, blue curves). For large $N \gg K$, the sampling error approaches values comparable to those obtained with private Gaussian noise (figure 5.5, blue vs. gray curves). For a broad range of N , the sampling error and the average shared-input correlation exhibit a similar trend ($\sim 1/N$).

5.3.3 Network-generated noise recovers sampling performance

In recurrent neural networks, inhibitory feedback naturally suppresses shared-input correlations through the emerging activity patterns (figure 5.2; Tetzlaff et al. (2012b)). Here we exploit this effect to minimize the detrimental influence of shared-input correlations that arise due to a limited number of noise sources. To this end, we replace the finite ensemble of independent stochastic sources with a recurrent network of deterministic units (figure 5.1, red). This noise network comprises excitatory and inhibitory binary units with a Heaviside activation function (see SI). Connections from the noise sources to the sampling network follow the same statistics as in the previous section. The connectivity within the noise network is random, sparse and homogeneous (Vreeswijk and Sompolinsky, 1996b; Renart et al., 2010b; Helias, Tetzlaff,

and Diesmann, 2014b; Dahmen, Bos, and Helias, 2016). To achieve optimal suppression of shared-input correlations, connectivity within the noise network needs to be statistically identical to the connectivity between the noise network and the sampling network. As discussed above, the additional contribution to the input fields h'_i of neurons in the sampling network can be approximated by a normal distribution $\mathcal{N}(\mu, \sigma^2)$. However, an additional term in the input variances arises from correlations between units in the noise network. We employ an iterative mean-field theory approach to estimate the average activities and average correlations in the noise network from the statistics of the connectivity. From these, μ and σ can be determined (see SI).

Using a recurrent network for noise generation considerably decreases the sampling error compared to the error obtained with a finite number of independent sources, even if the shared-input correlations are substantial (figure 5.4, red vs. blue line). Over a large range of noise-network sizes N , the input correlation and, hence the sampling error, are significantly reduced (figure 5.5, red vs. blue). In this range, the sampling error is comparable to the error obtained with private Gaussian noise and almost independent of N (figure 5.5, red vs. gray). Only if the noise network becomes too dense ($K \approx N$), its dynamics lock into a fixed point and the sampling performance breaks down.

5.3.4 Deterministic recurrent neural networks as a noise source for a generative model of handwritten digits

All realizations within the ensemble of unspecific, randomly generated sampling networks considered so far exhibit consistent performance characteristics (cf. narrow error bands in figure 5.4 and figure 5.5). We obtain similar behavior for a sampling network where the weights and biases are not chosen randomly but trained for a specific task – the generation of handwritten digits with imbalanced class frequencies (see SI): (i) networks with private external noise perform close to optimal, (ii) shared noise correlations impair network performance, and (iii) the performance is restored by employing a recurrent network for noise generation (figure 5.6). We thereby demonstrate that deterministic recurrent neural networks qualify as a suitable noise source for practical applications of stochastic neural networks.

5.3.5 Shared-input correlations impair network performance for high-entropy tasks

The dynamics of a BM representing a high-entropy distribution take place on a flat energy landscape with shallow minima. Here, the sampling process is sensitive to perturbations in statistical dependencies, such as those caused by shared-input correlations. In contrast, the sampling dynamics in BMs representing low-entropy distributions with pronounced peaks are dominated by deep minima in the energy landscape. In this case, noise correlations have little effect.

We systematically vary the entropy of the target distribution by changing the inverse temperature β in a BM and adjusting the relative noise strength in the other cases accordingly. Since β always appears as a multiplicative factor in front of weights and biases, this is equivalent to scaling weights and biases globally. For small entropies, the sampling error for shared and network noise is comparable to the error obtained with private noise, despite substantial shared-input correlations (figure 5.7). Consistent with the intuition provided above, the sampling error for

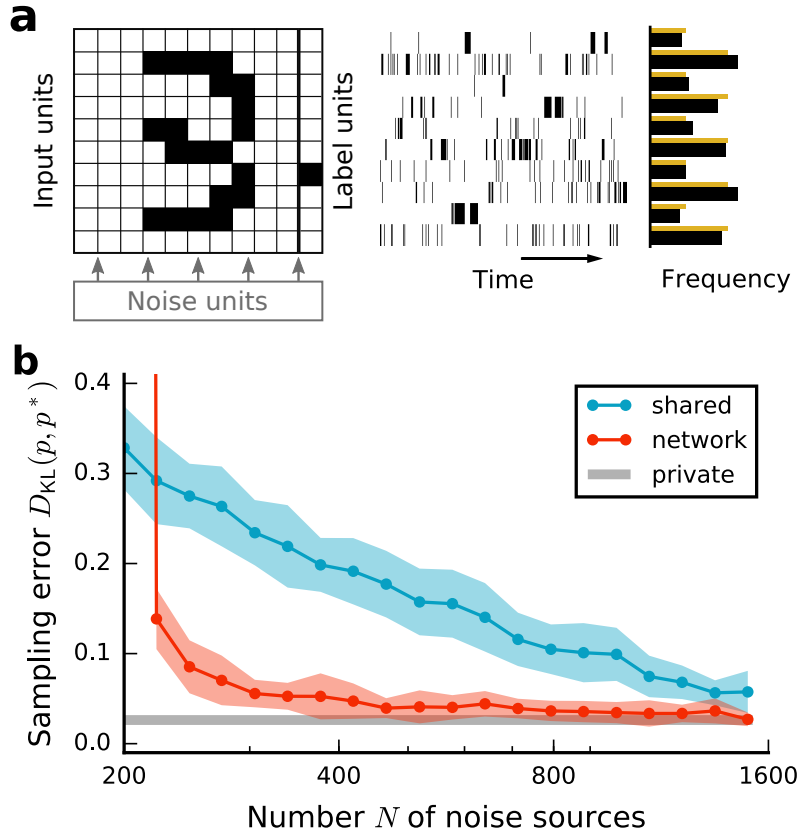


FIGURE 5.6: Performance of a generative network trained on an imbalanced subset of the MNIST dataset for different noise sources. (a) Left: Sketch of the network consisting of input and label units and external noise units. Right: Network activity and trial-averaged relative activity of label units for the intrinsic case (black) and target distribution (yellow), with even digits occurring twice as often as odd digits. (b) Sampling error $D_{KL}(p, p^*)$ between the empirical state distribution p of label units and the state distribution p^* of label units generated by a BM as a function of the number N of noise sources for shared and network case. Error bands indicate mean $\pm$ SEM over 20 trials with different initial conditions and noise realizations.

shared noise increases significantly with increasing entropy, whereas in the other cases it remains low.

We conclude that, in general, the effect of shared-noise correlations on the functional performance of sampling networks depends on the entropy of the target distribution. For high-entropy tasks, such as pattern generation, shared-input correlations can be highly detrimental. For low-entropy tasks, such as pattern classification, they play a less significant role. However, independent of the entropy of the task, functional performance for network-generated noise is close to optimal.

5.3.6 Small recurrent networks provide large sampling networks with noise

To achieve a good sampling performance, both the number N of noise sources as well as the number K of noise inputs per functional unit need to be sufficiently large (figure 5.5). Therefore, a certain minimal amount of resources have to be reserved for noise generation. We show here that once these resources are allocated, small recurrent networks can provide noise for large sampling networks without sacrificing

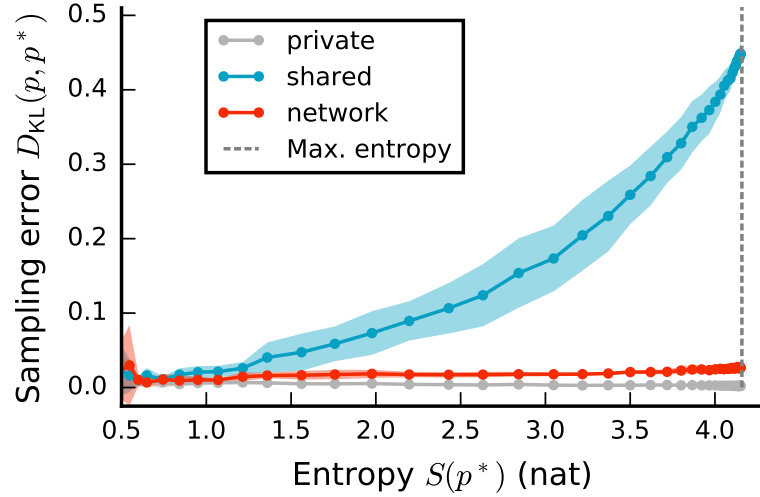


FIGURE 5.7: Sampling error $D_{\text{KL}}(p, p^*)$ as a function of the entropy S of the target distribution. Error bands indicate mean $\pm$ SEM over 5 random network realizations. Vertical dashed dark gray line indicates entropy of uniform distribution, i.e., maximal entropy. Sampling duration $T = 10^5$ ms. Remaining parameters as in figure 5.4.

computational performance. Note that, trivially, a single noise network can supply an arbitrary number of independent functional networks with noise.

Here, we vary the size of the sampling network M , while keeping N fixed. To compare the performances for different M , we observe only a fixed number m of neurons in the sampling network. By increasing M while keeping all other parameters constant, the entropy of the marginal distribution over the subset of observed neurons changes (see SI), thereby influencing the sampling performance in the presence of shared-noise correlations. Here we keep the entropy of the marginal target distribution approximately constant by scaling the weights in the sampling network with $1/\sqrt{M}$ (figure 5.8 inset, gray curve; see also Vreeswijk and Sompolinsky (1998), Renart et al. (2010b), and Helias, Tetzlaff, and Diesmann (2014b)).

In the presence of private noise, the sampling error is small and independent of M (figure 5.8). As before, the performance is considerably impaired for shared noise. Here, we observe a decrease in the error for larger sampling networks, which cannot be traced back to a change in entropy, by virtue of the experimental setup. Instead, the reason is that sampling networks with net recurrent inhibition, an increase in M strengthens, on average, the negative feedback in the sampling network, thereby causing a more efficient suppression of external correlations (Helias, Tetzlaff, and Diesmann, 2014b). Still, even for large M , the error remains significantly larger than the one obtained with private noise. For network noise, in contrast, it is almost as small as for private noise, and independent of M . Qualitatively similar findings are also obtained without scaling synaptic weights unless the entropy of the target distribution becomes too small (see SI).

5.3.7 Networks of spiking neurons implement neural sampling without noise

The results so far rest on networks of binary model neurons. Their dynamics are well understood (Ginzburg and Sompolinsky, 1994; Vreeswijk and Sompolinsky, 1998;

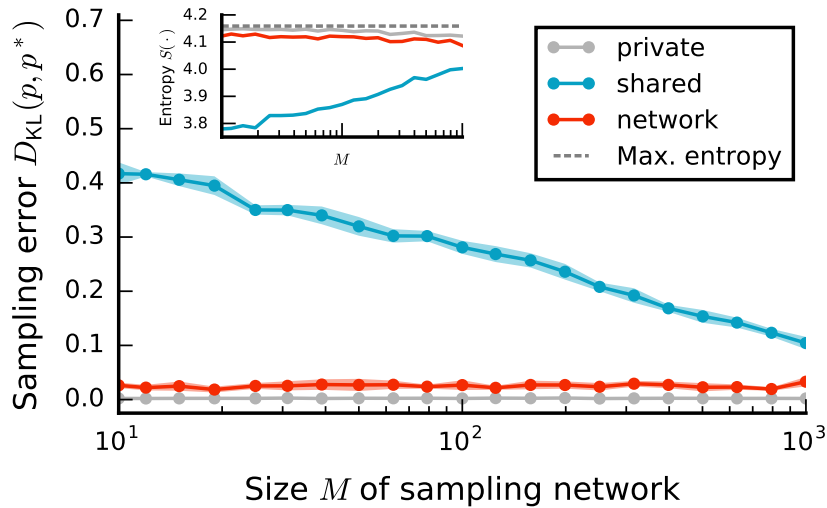


FIGURE 5.8: Sampling error $D_{\text{KL}}(p, p^*)$ as a function of the sampling-network size M . Error bands indicate mean $\pm$ SEM over 5 random network realizations. Inset: Entropy of the sampled state distribution p as a function of M . Horizontal dashed dark gray line indicates entropy of uniform distribution, i.e., maximal entropy. Average weight in sampling networks: $\mu_{\text{BM}} = -0.15/\sqrt{M}$. Sampling duration $T = 10^5$ ms. Remaining parameters as in figure 5.4.

Renart et al., 2010b; Helias, Tetzlaff, and Diesmann, 2014b; Dahmen, Bos, and Helias, 2016), and their mathematical tractability simplifies the calibration of sampling-network parameters for network-generated noise. Physical neurons in nature and in neuromorphic hardware are, however, predominantly spiking (Nawrocki, Voyles, and Shaheen, 2016; Furber, 2016). Indeed, networks of spiking neurons with private external noise can represent arbitrary target distributions ((Petrovici et al., 2016; Probst et al., 2015), see also gray curve in figure 5.9), by an appropriate translation of parameters from binary units to spiking neurons (see SI). Consistent with our results on binary networks the sampling performance decreases in the presence of shared-noise correlations, but recovers for noise provided by a recurrent network of spiking neurons (figure 5.9). Similar to binary networks, a minimal noise-network size N ensures an asynchronous activity regime, a prerequisite for good sampling performance. Spiking noise networks that are too densely connected ($K/N \rightarrow 1$) tend to synchronize, causing large sampling errors (see red curve in figure 5.9 for $N < 1500$).

5.4 Discussion

Consistent with the high variability in the activity of biological neural networks (Faisal, Selen, and Wolpert, 2008b), many models of high-level brain function rely on the presence of some form of stochasticity. We propose that additive input from deterministic recurrent neural networks serves as a well controllable source of noise for functional network models. In this chapter, we demonstrate that networks of deterministic units with input from such noise-generating networks can approximate a large variety of target distributions and perform well in generative tasks. This scheme covers both networks of binary and networks of spiking neurons, and leads to an economic resource usage in biological and artificial neuromorphic systems.

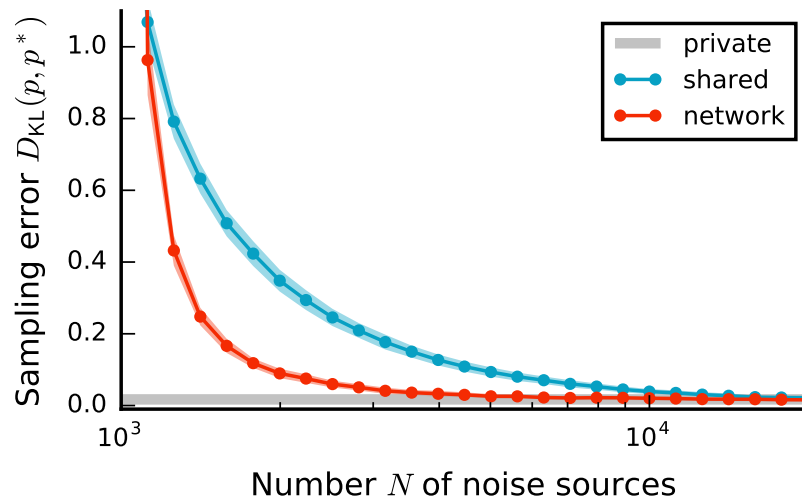


FIGURE 5.9: Sampling error $D_{\text{KL}}(p, p^*)$ between the empirical state distribution p of a sampling network of spiking neurons and the state distribution p^* generated by a BM as a function of the number N of noise sources for shared and network case. Error bands indicate mean $\pm$ SEM over 10 random network realizations. See SI for model parameters.

For conceptual simplicity, this study segregates a neuronal network into a functional and a noise-generating module. In biological substrates, these two modules may be intermingled. Alternatively, a brain may use one sampling network as the noise source for another. In this view, one network's function is another network's noise.

We show that shared-noise correlations can be highly detrimental for sampling from given target distributions. Generating noise with recurrent neural networks overcomes this problem by exploiting active decorrelation in networks with inhibitory feedback (Renart et al., 2010b; Tetzlaff et al., 2012b). As an alternative solution, the effect of shared-input correlations could be mitigated by training functional network models in the presence of these correlations (Bytschok et al., 2017). However, this approach is specific to particular network models. Moreover, it prohibits porting of models between different substrates. Networks previously trained under specific noise conditions will not perform well in the presence of noise with a different correlation structure. Our approach, in contrast, constitutes a general-purpose solution which can also be employed for models that cannot easily be adapted to the noise statistics, such as bottom-up biophysical neural-network models (e.g., Potjans and Diesmann, 2012).

In biological neural networks, the probabilistic gating of ion channels in the cell membrane (White, Rubinstein, and Kay, 2000) and the variability in synaptic transmission (Branco and Staras, 2009) constitute alternative potential sources of stochasticity. However, for the majority of stochastic network models, ion-channel noise is too small to be relevant: in the absence of (evoked or spontaneous) synaptic input, fluctuations in membrane potentials recorded *in vitro* are in the μV range and hence negligible compared to the mV fluctuations necessary to support sampling-based approaches (Petrovici et al., 2016). Synaptic stochasticity comes in two distinct forms: spontaneous release and variability in evoked postsynaptic response amplitudes. The rate of spontaneous synaptic events measured at the soma of the target neuron is in the range of a few events/second (Hardingham and Larkman, 1998;

Locke, Vautrin, and Highstein, 1999). The resulting fluctuations in the input associated with spontaneous release are therefore negligible. The variability in postsynaptic response amplitudes, in contrast, is substantial (Branco and Staras, 2009), and has often been suggested as a plausible noise resource for computations in neural circuits (Levy and Baxter, 2002; Rosenbaum, Rubin, and Doiron, 2012; Maass, 2014; Kappel et al., 2015; Neftci et al., 2016; Muller and Indiveri, 2017). Due to its multiplicative, state-dependent nature, this form of noise is fundamentally different from the noise usually employed in sampling models. Its role for approximate inference in neural substrates remains unclear.

In neuromorphic-hardware systems, there exist a few alternative approaches to the generation of uncorrelated noise for stochastic network models, such as exploiting thermal noise and trial-to-trial fluctuations in neuron parameters (see, e.g., Hamid, Tang, and Murray, 2011; Binas, Indiveri, and Pfeiffer, 2016; Sengupta et al., 2016). However, hardware systems need to be specifically designed for these approaches and sacrifice chip area that otherwise could be used to house neurons and synapses. The solution proposed in this chapter does not require specific hardware components for noise generation. It solely relies on the capability of emulating recurrent neural networks, the functionality most neuromorphic-hardware systems are designed for.

In contrast to numerical simulations, in which all parameters of a network can be finely controlled, neuromorphic systems that employ analog components show strong spatial and temporal heterogeneities. So far it is unclear in how far these properties influence the correlation structure of recurrent networks. In the next chapter we explore in how far the suppression of shared-input correlations through activity correlations arising from strong inhibitory effect is present in network emulated in the mixed-signal system Spikey (Pfeil et al., 2013c).

Chapter 6

The effect of heterogeneity on decorrelation mechanisms in spiking neural networks: a neuromorphic-hardware study

This chapter is based on the following publication:

Thomas Pfeil, Jakob Jordan, Tom Tetzlaff, Andreas Grübl, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier (2016). “Effect of Heterogeneity on Decorrelation Mechanisms in Spiking Neural Networks: A Neuromorphic-Hardware Study”. In: *Phys. Rev. X* 6.2. 021023

Equal contribution by Thomas Pfeil & Jakob Jordan

Author contributions:

Thomas Pfeil and the author performed the work concertedly with equal shares on almost all parts of the study. The recording of data from the neuromorphic chip, was done by Thomas Pfeil, while the software simulations were performed by the author. Tom Tetzlaff supervised the work and all authors contributed to the writing of the manuscript.

6.1 Introduction

Dynamical systems in nature often exhibit a remarkable degree of diversity, specialization or anticorrelation across their components, despite equalizing factors such as common input or homogeneity in component and interaction parameters. In many cases, these observations can be explained by the effect of negative feedback. Cell differentiation caused by lateral inhibition (Gierer and Meinhardt, 1974), formation of new species driven by competition (Rosenzweig, 1978) or antiferromagnetism (Neel, 1952) constitute just a few examples. In recurrent neuronal networks, inhibitory feedback constitutes a powerful decorrelation mechanism which allows different neurons to respond nearly independently, even if they are driven by largely overlapping local or external inputs (Renart et al., 2010a; Tetzlaff et al., 2012a; Helias, Tetzlaff, and Diesmann, 2014a). Decorrelation by negative feedback hence implements an efficient form of redundancy reduction. In biological systems, it may serve similar purposes as decorrelation in technical applications, where it is used in data compression (e.g., principal-component analysis (Pearson, 1901)), crosstalk reduction (e.g., in digital signal processing (Ginis and Peng, 2006)), echo suppression (e.g., in acoustics (Benesty et al., 2001)) or random-number generation in hardware (Trichina et al., 2001). Moreover, inhibitory feedback suppresses “quantization noise” at low frequencies and can thereby increase the dynamical range and signal-to-noise ratio for the encoding of analog signals in the spiking activity of recurrent neural networks (Mar et al., 1999). It is tempting to exploit these mechanisms in synthetic, neurally inspired architectures such as analog neuromorphic hardware.

Analog neuromorphic hardware mimics properties of biological neural systems using physical models of neurons and synapses (capacitors, for example, emulate insulating cell membranes) (Mead, 1990; Indiveri et al., 2011a). The temporal evolution of the analog circuits represents a solution to the corresponding model equations. In consequence, neural-network emulations on analog neuromorphic hardware are massively parallel, extremely fast and energy efficient. Analog neuromorphic devices are therefore highly attractive as tools for neuroscientific research, e.g., for the investigation of learning on long time scales, and technical applications (SenseMaker, 2009; FACETS, 2010; BrainScaleS, 2014; Human Brain Project, 2014). A biologically inspired neural network (olfactory system of insects) performing rapid online data (odor) classification, for example, has recently been successfully implemented on the analog neuromorphic hardware system *Spikey* (Pfeil et al., 2013b; Schmuker, Pfeil, and Nawrot, 2014). In this application, decorrelation by inhibition is an essential ingredient to guarantee high classification performance. The suppression of quantization noise by inhibitory feedback (Mar et al., 1999) has been used as a means of noise shaping in several neuromorphic-hardware applications, aiming at the construction of biologically inspired ultra-low power analog-to-digital converters (Marienborg, Lande, and Hovin, 2002; Kikombo et al., 2009).

For the functional performance of neuronal architectures, the level of correlations between the activities of individual neurons is often pivotal. Whether such correlations are beneficial or not is context dependent. A number of previous studies emphasize a functional benefit of certain types of correlation for encoding/decoding of information in/from populations of neurons (Ecker et al., 2011; Averbeck, Latham, and Pouget, 2006; Moreno-Bote et al., 2014), information transmission (Fries, 2005; Abeles, 1991; Diesmann, Gewaltig, and Aertsen, 1999), robustness against noise (Tabareau, Slotine, and Pham, 2010), or gain control of postsynaptic neurons (Salinas and Sejnowski, 2001). Other studies argue that positive cross-correlations are detrimental as they decrease the precision or sparseness of population codes (Shadlen

and Newsome, 1998; Abbott and Dayan, 1999; Tripp and Eliasmith, 2007; Schmuker and Schneider, 2007; Schmuker, Pfeil, and Nawrot, 2014). Cohen and Maunsell (2009), for example, have shown that decreased spike-train correlations in macaque visual area V4 are accompanied by increased behavioral performance in an orientation change-detection task. Depending on the similarity between the trial-averaged responses of different neurons to external stimuli (signal correlation), noise correlations (correlations not explained by signal correlations) can either increase or decrease the amount of information that can be encoded in or decoded from a population of neurons. In populations of neurons with high signal correlation, vanishing or even negative noise correlations are desirable to improve the population code (Averbeck, Latham, and Pouget, 2006).

In finite neural networks, an inevitable source of correlated neural activity is common presynaptic input, shared by multiple postsynaptic neurons. In network models and in-vivo recordings, however, pairwise correlations in the activity of neighboring neurons have been found to be substantially smaller than expected given the amount of shared input (Tetzlaff et al., 2004; Kriener et al., 2008; Tetzlaff et al., 2008; Ecker et al., 2010; Renart et al., 2010a; Tetzlaff et al., 2012a). In several studies, this observation has been explained by inhibitory coupling. While Ly, Middleton, and Doiron (2012) and Middleton et al. (2012) primarily focused on the effect of feedforward inhibition, Renart et al. (2010a), Wiechert et al. (2010), and Tetzlaff et al. (2012a) attributed the smallness of correlations to an active decorrelation of neural activity by inhibitory feedback. The mechanism underlying this active decorrelation has already been described by Mar et al. (1999). In this study, the authors focused on the suppression of low-frequency fluctuations of the population firing rate by recurrent dynamics. As the amplitude of population-rate fluctuations is directly linked to pair-wise correlations (see, e.g., Harris and Thiele (2011)), the effect described in Mar et al. (1999) corresponds to a suppression of pairwise correlations in the spiking activity. The theory underlying decorrelation by inhibitory feedback suggests the effect to be general: Decorrelation should be observable in any system with sufficiently strong inhibitory feedback, irrespective of the details of the network structure and the cell and synapse properties. For networks of spiking neurons, however, the effect has so far been explicitly demonstrated only for the homogeneous case, where all neurons have identical properties, receive (approximately) the same number of inputs, and, hence, fire at about the same rate (Renart et al., 2010a; Tetzlaff et al., 2012a). Moreover, the sub-threshold dynamics of individual neurons was assumed to be linear.

Biological neuronal networks typically exhibit broad, heavy-tailed firing-rate distributions (Griffith and Horn, 1966; Koch and Fuster, 1989; Shafi et al., 2007; Hromadka, DeWeese, and Zador, 2008; O'Connor et al., 2010; Roxin et al., 2011; Buzsaki and Mizuseki, 2014), indicating a high degree of heterogeneity, e.g., in synaptic weights (Song et al., 2005; Lefort et al., 2009; Koulakov, Hromadka, and Zador, 2009; Avermann et al., 2012; Ikegaya et al., 2013), in-degrees (Roxin, 2011) or time constants (Kuhn, Aertsen, and Rotter, 2004; Roxin et al., 2011). The same holds for neural networks implemented on analog neuromorphic hardware. All analog circuits suffer from device variations caused by unavoidable variability in the manufacturing process. Neurons and synapses implemented in analog neuromorphic hardware therefore exhibit heterogeneous response properties, similar to their biological counterparts (Stein, Gossen, and Jones, 2005; Marder and Goaillard, 2006). To understand the dynamics and function of recurrent neural networks in both biological and synthetic substrates, it is therefore essential to account for such heterogeneities.

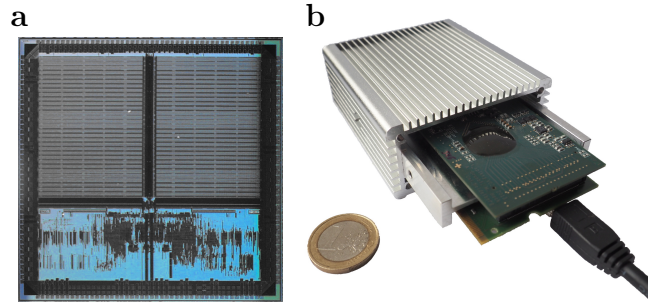


FIGURE 6.1: The neuromorphic hardware system *Spikey*. **(a)** Photograph of the *Spikey* chip (size $5 \times 5 \text{ mm}^2$). It comprises analog circuits of 384 neurons and 98304 synapses, is highly configurable and emulates neural-network dynamics with a speed-up of 10^4 with respect to biological real-time. **(b)** Photograph of the partly cased *Spikey* system, carrying the *Spikey* chip (covered by a black round seal) and conventional memory. The system is connected to the host computer via USB 2.0, consumes 6 W of power in total and less than 1 nJ per synaptic transmission (see appendix B.3).

Previous work on recurrent neural networks has shown that heterogeneity in single-neuron properties or connectivity broadens the distribution of firing rates (Van Vreeswijk and Sompolinsky, 1998; Roxin et al., 2011) and affects the stability of asynchronous or oscillatory states (Tsodyks, Mitkov, and Sompolinsky, 1993; Golomb and Rinzel, 1993; Neltner et al., 2000; Denker et al., 2004; Roxin, 2011; Mejias and Longtin, 2012). A number of studies pointed at a potential benefit of heterogeneity for the information-processing capabilities of neural networks (Stocks, 2000; Shamir and Sompolinsky, 2006; Chelaru and Dragoi, 2008; Osborne et al., 2008; Padmanabhan and Urban, 2010; Marsat and Maler, 2010; Holmstrom et al., 2010; Yim, Aertsen, and Rotter, 2013; Mejias and Longtin, 2012; Lengler, Jug, and Steger, 2013; Mejias and Longtin, 2014; Tripathy et al., 2013). The effect of heterogeneity on correlations in the activity of recurrent networks of spiking neurons, however, remains unclear. Padmanabhan and Urban (2010) have shown that the responses of a population of unconnected neurons are decorrelated by heterogeneity in the neuronal response properties. These results are supported by the subsequent theoretical analysis in Yim, Aertsen, and Rotter (2013). In the following, we refer to this type of decorrelation by heterogeneity as *feedforward decorrelation*. It does not account for the effect of the recurrent network dynamics. Active decorrelation due to inhibitory feedback (see above; Renart et al., 2010a; Tetzlaff et al., 2012a), in contrast, constitutes a very different mechanism. The effect of heterogeneity on this *feedback decorrelation* has lately been studied by Bernacchia and Wang (2013) in the framework of a recurrent network of linear firing-rate neurons. In this setup, correlations are suppressed by heterogeneity in the network connectivity (distributions of coupling strengths or random dilution of connectivity). It remains unclear, however, whether this holds true for networks of (nonlinear) spiking neurons.

In this study, we investigate the impact of heterogeneity on input and output correlations in the asynchronous regime of sparse networks of leaky integrate-and-fire (LIF) neurons with conductance-based synapses. Emulation of the networks on the analog neuromorphic hardware system *Spikey* (figure 6.1) (Schemmel et al., 2006; Pfeil et al., 2013b) enable us to investigate the impact of substrate specific properties on the network dynamics. Insights about the interplay between features of the computing substrate and network dynamics are a necessary prerequisite for the development of algorithms that exploit the benefits of analog neuromorphic systems

at best. The configurability of this system (Pfeil et al., 2013b) enables us to systematically vary the level of heterogeneity, and to disentangle the effects of heterogeneity on feedforward and feedback decorrelation (see above). For simplicity, we focus on purely inhibitory networks, thereby emphasizing that active decorrelation by inhibitory feedback does not rely on a dynamical balance between excitation and inhibition (Tetzlaff et al., 2012a; Helias, Tetzlaff, and Diesmann, 2014a). We show that decorrelation by inhibitory feedback is effective even in highly heterogeneous networks with broad distributions of firing rates (section 6.3.1). Increasing the level of heterogeneity has two effects: Feedforward decorrelation is enhanced, feedback decorrelation is impaired. Due to the latter, the overall input and output correlations do not necessarily become smaller with increasing heterogeneity. They can even increase (section 6.3.2).

Note that results from specific network emulations on hardware do not directly translate to those obtained by simulations on conventional computers, because the dynamics, parametrization and interplay of analog circuits is very complex and difficult to reproduce with classical simulations. If simplified models for spatial and temporal variability are considered in software simulations, however, emulation results can be reproduced qualitatively, thereby verifying the design of the hardware system. While our hardware system is designed to physically implement biologically realistic neural algorithms in a fast and energy-efficient way, software simulations are used as a complementary tool to isolate, verify and investigate different hardware features, such as spatial and temporal parameter variations. Due to the limited access and configurability of network parameters, this would be difficult to achieve with hardware studies alone. In analogy to the necessity of performing experiments on biological neural systems to verify assumptions made in computational neuroscience, actual emulations on neuromorphic hardware are essential to understand its properties and develop efficient neural algorithms for these devices. The fact that our main findings hold true for both emulations on hardware and simulations with software, and that they can be distilled to simple linear models, support their broad relevance and robustness.

6.2 Methods

6.2.1 Network model

Details on the network, neuron and synapse model are provided in table B.1. Parameter values are given in table B.2. Briefly: We consider a purely inhibitory, sparse network of N ($N = 192$, unless stated otherwise) LIF neurons with conductance-based synapses. Each neuron receives input from a fixed number $K = 15$ of randomly chosen presynaptic sources, independently of the network size N . Self-connections and multiple connections between neurons are excluded. Resting potentials E_1 are set above the firing thresholds Θ (equivalent to applying a constant supra-threshold input current). We thereby ensure autonomous firing in the absence of any further external input. Due to temporal noise, the initial conditions are essentially random.

6.2.2 Network emulations on the neuromorphic-hardware system *Spikey*

The *Spikey* chip (figure 6.1) consists of physical models of LIF neurons and conductance-based synapses with exponentially decaying dynamics (for details, see table B.1). The emergent dynamics of these physical models represents a solution for the model equations of neurons and synapses in continuous time, in parallel for all units. In

contrast, in classical simulations on von-Neumann architectures, model equations are solved by step-wise numerical integration, where parallelization is limited by the available number of processor cores. To emphasize the difference between simulations using software and simulations using physical models, the term *emulation* is used for the latter (Pfeil et al., 2013b).

The response properties of physical neurons and synapses vary across the chip due to unavoidable variations in the production process that manifest in a spatially disordered pattern (*fixed-pattern noise*). In contrast to the approximately static fixed-pattern noise, *temporal noise*, including electronic noise and transient experiment conditions (e.g., chip temperature), impairs the reproducibility of emulations. In general, two network emulations with identical configuration and stimulation do not result in identical network activity. Both fixed-pattern and temporal noise need to be taken into account when developing models for analog neuromorphic hardware.

The key features of the *Spikey* chip are the high acceleration and configurability of the analog network implementation. Some network parameters, e.g., synaptic weights and leak conductances, are configurable for each unit, while other parameters are shared for several units (for details see Pfeil et al. (2013b)). The hardware system is optimized for spike in- and output and allows to record the membrane potential of one (arbitrarily chosen) neuron with a sampling frequency of 96 MHz in hardware time. On the *Spikey* chip, capacitances are smaller and conductances are much higher than in biological nervous systems. In consequence, networks on the *Spikey* chip are emulated with a speed-up of approximately 10^4 with respect to biological real-time. Due to this high acceleration of the neuromorphic chip, the data bandwidth of the connection between the neuromorphic system and the host computer is not sufficient to communicate with the chip in real time. Consequently, input and output spikes (for stimulation and from recordings, respectively) are buffered in a local memory next to the chip. The high acceleration of the *Spikey* chip allows most of the transistors to operate outside of weak inversion, thereby reducing the effect of transistor variations and minimizing fixed-pattern noise.

In contrast to such accelerated systems, most other configurable, analog neuromorphic substrates are designed for real-time emulations at very low power consumption (Badoni et al., 2006; Hafliger, 2007; Vogelstein et al., 2007; Indiveri, Chicca, and Douglas, 2009; Serrano-Gotarredona et al., 2009; Brink et al., 2013; Benjamin et al., 2014) and implement fewer, but more complex, neurons (Renaud et al., 2010; Yu and Cauwenberghs, 2010).

Access to the *Spikey* system is encapsulated by the simulator-independent language `PyNN` (Davison et al., 2009; Brüderle et al., 2009), providing a stable and user-friendly interface. `PyNN` integrates the hardware into the computational neuroscience tool chain and has facilitated the implementation of several network models on the *Spikey* chip (Kaplan et al., 2009; Bill et al., 2010; Brüderle et al., 2010; Pfeil et al., 2013b; Schmuker, Pfeil, and Nawrot, 2014).

On the *Spikey* system, a spiking neural network is emulated as follows (figure 6.2a): First, the network described in `PyNN` is mapped to the *Spikey* chip, i.e., neurons and synapses are allocated and parametrized. Second, input spikes, if available, are prepared on the host computer and transferred to the local memory on the hardware system. Third, the emulation is triggered and available input spikes are generated. Output spikes and membrane data are recorded to local memory. Last, spike and membrane data are transferred to the host computer and scaled back into the biological domain of the `PyNN` model description.

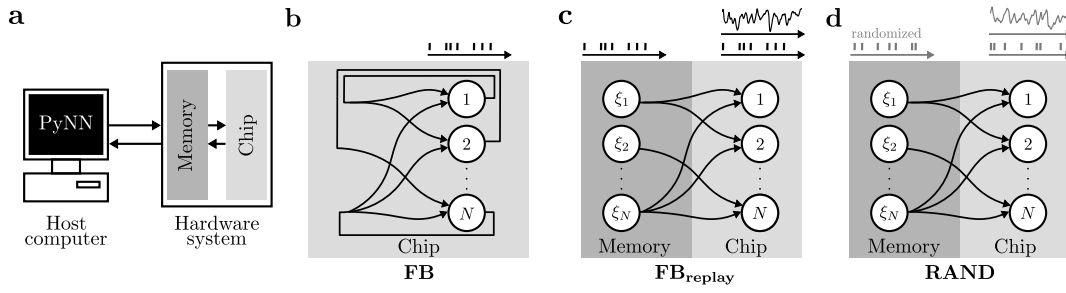


FIGURE 6.2: Experimental setup. **(a)** Data flow of the *Spikey* system. For details see section 6.2.2. **(b)** Network with on-chip feedback connections (FB). Spikes from all neurons are recorded to the local memory. **(c)** Spikes of the FB network in (b) replayed from memory via off-chip spike sources ξ_i to neurons i (FB_{replay}). Spike times of ξ_i correspond to those recorded from neuron i in (b). Spikes from all neurons or the free membrane potential of one selected neuron are recorded. **(d)** Like (c), but spike times from (b) are randomized for each source (RAND).

For consistency with the model description and simplified comparison to the existing literature, all hardware times and all hardware voltages are expressed in terms of the quantities they represent in the neurobiological model, throughout this study.

6.2.3 Experimental setup

To differentiate and compare the effects of shared inputs and feedback connections on correlations, we investigate two different emulation scenarios: First, we emulate networks with intact feedback (FB, figure 6.2b), and second, the contribution of shared input is isolated by *randomizing* the temporal order of this feedback (RAND, figure 6.2d).

In the RAND scenario, the inputs of neurons are decoupled from their outputs. Spatio-temporal correlations in presynaptic spike trains are removed by randomizing the presynaptic spike times.

Input correlations between neurons are measured via their *free membrane potential*, i.e., the membrane potential with disabled spiking mechanism (technically, the threshold is set very high). Because membrane potential traces can be recorded in the hardware only one at a time, traces are obtained consecutively, while repeatedly *replaying* the previously recorded activity of the FB network to a population of unconnected neurons of equal size. We keep the connectivity the same, and hence each neuron receives the same number of spikes as in the recurrent network during the whole emulation, either without (FB_{replay}, figure 6.2c) or with randomization of presynaptic spike times (RAND, figure 6.2d), respectively. To preserve the fixed pattern of variability of synaptic weights in hardware, the same hardware synapses are used for each connection in both scenarios. If network dynamics were reproduced perfectly, membrane potential traces and spike times would be identical in the FB and FB_{replay} cases (see also section 6.2.4).

Drawing two different network realizations (i.e., the connectivity matrix) results in the allocation of different hardware synapses, and, due to fixed-pattern noise, in different values of synaptic weights. To average over this variability, throughout this study, emulation results are averaged over $M = 100$ network realizations, if not stated otherwise.

6.2.4 Reproducibility of hardware emulations

Since the initial conditions of the recurrent network on hardware are undefined, consecutive emulations of the FB network result in different network activities. In the RAND and FB_{replay} case, however, the input of neurons is decoupled from their output. Although unavoidable temporal noise is present, the system's state-space trajectory returns to the trajectory of the previously recorded FB case. A certain degree of reproducibility is required for two reasons: First, the investigated effect of decorrelation by inhibitory feedback requires a precise relation between spike input and output. Thus our method of replacing the feedback loop by replay is only valid if temporal noise does not substantially corrupt this relationship. Second, to record the membrane potentials of all neurons, as if recorded at once, neuron dynamics have to be reasonably similar in consecutive emulations.

We measure the reproducibility of neuron dynamics by comparing consecutive emulations with identical configuration, i.e., connectivity and stimulation. For this purpose the spiking activity of a FB network is first recorded (figure 6.2b) and then repeatedly replayed (figure 6.2c). Reproducibility is quantified by the correlations (κ_X in table B.3) of free membrane potential traces and output spike trains obtained for individual neurons in $L = 25$ different trials.

Free membrane potentials are reproduced quite well, while spike trains show larger deviations across trials (figure 6.3). Small deviations in the membrane potential (figure 6.3b) are amplified by the thresholding procedure (Mainen and Sejnowski, 1995b; De la Rocha et al., 2007; Wiechert et al., 2010) and can lead to large differences between spike trains (figure 6.3c). Consequently, measures based on data of several consecutive replays are more precise for membrane potentials than for spike trains. Nevertheless, results have to be interpreted with care in both cases.

6.2.5 Calibration

The heterogeneity of the *Spikey* hardware is adjusted by calibrating the leak conductance¹ for each individual neuron, compensating for fixed-pattern noise of neuron parameters. To this end, a population of unconnected neurons is driven by a constant supra-threshold current and the time-averaged population activity $\bar{\nu}$ is measured. Then, we applied the bisection method (Press et al., 2007) to adjust the leak conductance g_l of each neuron, such that the neuron's firing rate matches the target rate $\bar{\nu}$. This results in calibration values b for the leak conductance $g_l = g_{l,0}(1 + b)$, where $g_{l,0}$ is the leak conductance before calibration. Because emulations on hardware are not perfectly reproducible, more precise calibration was achieved by evaluating the median over 25 identically configured trials instead of single trials. Furthermore, the bisection method was modified for noisy systems (for details, see appendix B.4).

Intermediate calibration states are obtained by linearly scaling the full calibration:

$$g_l = g_{l,0}(1 + (1 - a)b) \quad . \quad (6.1)$$

The heterogeneity a is chosen in $[0, 1]$ for calibrations between the uncalibrated ($a = 1$) and calibrated state ($a = 0$). In the following, the fully calibrated chip ($a = 0$) is used, if not stated otherwise.

¹since capacitances and potentials cannot be configured individually for each hardware neuron (Pfeil et al., 2013b)

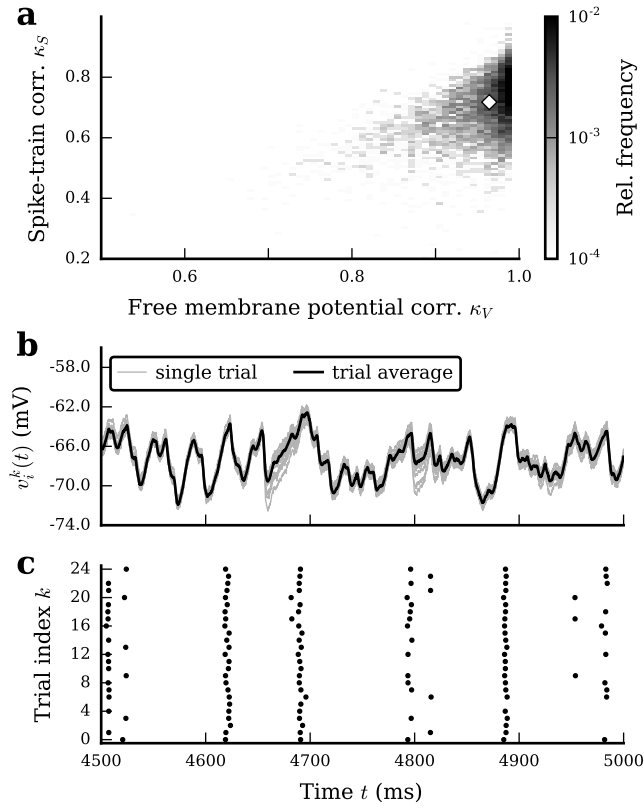


FIGURE 6.3: Reproducibility of free membrane potentials and spiking activity in the $\text{FB}_{\text{replay}}$ case. **(a)** Low-frequency coherence κ_V and κ_S of free membrane potentials $v_i^k(t)$ and $v_i^l(t)$ and binned spike trains $s_i^k(t)$ and $s_i^l(t)$, respectively, for each neuron i averaged over $L = 25$ trials k, l with $k \neq l$, for $M = 50$ different network realizations. The diamond marks the average across all neurons i and M network realizations ($\kappa_V = 0.96$, $\kappa_S = 0.72$). **(b)** Free single-trial membrane potentials $v_i^k(t)$ (gray) and average over trials $\frac{1}{L} \sum_{k=1}^L v_i^k(t)$ (black), and **(c)** spike density $\xi_i(t)$ of a single neuron i for $L = 25$ identical trials. The selected neuron i has membrane potential coherence and spike train coherence closest to the diamond in (a).

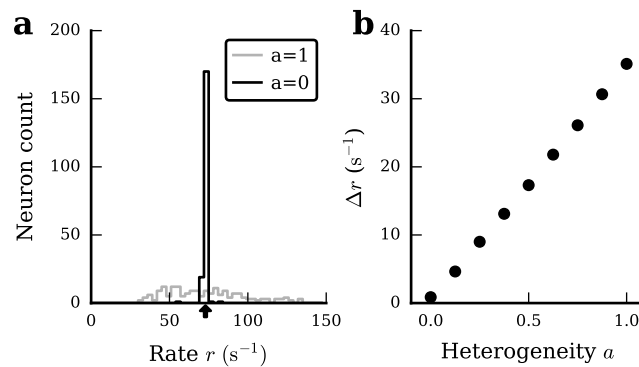


FIGURE 6.4: Calibration of the *Spikey* chip. **(a)** Histogram of firing rates ν for a population of unconnected neurons with supra-threshold input currents, before (gray) and after (black) calibration, each neuron averaged over $L = 100$ trials. The arrow denotes the target rate $\bar{\nu}$. **(b)** Difference $\Delta r = r_{P75} - r_{P25}$ of 75th and 25th percentile of the histograms in (a), as a function of network heterogeneity a (equation (6.1)). The mean firing rate over all values of a is $73.4(3) \text{ s}^{-1}$.

This calibration substantially narrows the distribution of firing rates compared to the uncalibrated state (figure 6.4). With respect to the stationary firing rate, variability on the neuron level is reduced from 35.1 to 0.9 s^{-1} .

Even in the fully calibrated state, leak conductances can still be widely distributed. Due to the chosen calibration procedure, they are likely to be correlated to other parameters that influence the neurons' response to a constant supra-threshold current after calibration. This mutual compensation can lead to similar phenomenology (here: firing rates) despite disparate parameter values, similar to what is observed in biology (Prinz, Bucher, and Marder, 2004). In addition to remaining variations in neuron parameters, synaptic parameters are significantly distributed (Pfeil et al., 2013a; Schmuker, Pfeil, and Nawrot, 2014).

6.2.6 Correlation measures

In the following, we introduce definitions used to analyze the recorded data. For clarity, all relevant equations and their parametrization are listed in table B.3 and B.4, respectively.

We quantify correlations of membrane potentials $v_i(t)$ and spike trains $s_i(t)$ by the population-averaged *low-frequency coherence* κ_V and κ_S , respectively. At frequency zero, the coherence corresponds to the normalized integral of the cross-covariance function, i.e., it measures correlations on all time scales. We define the low-frequency coherence κ_X , with $X \in \{S, V\}$, to be the average coherence over a frequency interval from 0.1 to 20 Hz . In this interval, the suppression of population-rate fluctuations in recurrent networks due to inhibitory feedback is most pronounced, and the coherence is approximately constant. Before calculating the coherence, we convolve the power- and cross-spectra with a rectangular window to average out random fluctuations. This measure, or a variant of it, is commonly used in the neuroscientific literature (Bair, Zohary, and Newsome, 2001; Kohn and Smith, 2005; Moreno-Bote and Parga, 2006; De la Rocha et al., 2007; Renart et al., 2010a; Tetzlaff et al., 2012a; Yim, Aertsen, and Rotter, 2013). We use the terms low-frequency coherence and correlation interchangeably.

Throughout this study, the term *input correlations* is used for correlations between free membrane potentials, and *output correlations* for correlations between spike trains. *Shared-input correlations* are membrane-potential correlations that are exclusively caused by overlapping presynaptic sources, ignoring possible correlations in the presynaptic activity. In homogeneous networks, the average pairwise shared-input correlation

$$\kappa_V = \frac{K}{N} \quad (6.2)$$

is given by the connectivity K/N (Tetzlaff et al., 2012a). In heterogeneous networks, shared-input correlations can be reduced. In the presence of heterogeneous synaptic weights, for example, the shared-input correlation

$$\kappa_V = \frac{1}{1 + CV_J^2} \frac{K}{N} \quad (6.3)$$

is decreased by a factor $1/(1 + CV_J^2)$, where CV_J denotes the coefficient of variation of the (non-zero) synaptic weights. Note, however, that heterogeneities which affect only the spike generation but not the integration of synaptic inputs, e.g. distributions of firing thresholds, have no effect on the shared-input correlation.

We assess the significance of correlations by comparing the results from emulations to correlations in surrogate data, in which we removed spatial correlations. For every neuron, we randomly shuffled bins of the membrane potential trace, and assigned a new timestamp uniformly drawn from the emulation interval to every spike, respectively. We thereby remove all spatio-temporal correlations between neurons recorded in parallel. By this procedure we create 100 surrogate trials, across which we calculate the average correlations and the standard error.

To quantify fluctuations in the population activity $\bar{s}$ (figure 6.5a–c, horizontal histograms) we compute the power spectrum $\bar{A}(f)$ of the population activity (figure 6.5e), which we scale with the duration T of the emulation. Consequently, the population power spectrum $\bar{A}(f)$, scaled by the population size, coincides with the time-averaged population activity $\bar{\nu}$ for high frequencies: $\lim_{f \rightarrow \infty} \frac{1}{N} \bar{A}(f) = \bar{\nu}$ (Tetzlaff et al., 2008).

As a measure of pairwise correlations in the time domain (figure 6.5d), we compute the population-averaged cross-correlation function $c(\tau)$ by Fourier transforming the population-averaged cross-spectrum $C(f)$ to time domain.

6.3 Results

In this study, we investigate the roles of shared input, feedback and heterogeneity on input and output correlations in random, sparse networks of inhibitory LIF neurons with conductance-based synapses (table B.1), implemented on the analog neuromorphic hardware chip *Spikey* (figure 6.1). Similarly to Tetzlaff et al. (2012a), we separate the contributions of shared input and feedback by studying different network scenarios (figure 6.2): In the FB case, we emulate the recurrent network with intact feedback loop (figure 6.2b) and record its spiking activity (figure 6.5a). In the FB_{replay} case (figure 6.2c), the feedback loop is cut and replaced by the activity recorded in the FB network. Ideally, the input to each neuron in the FB_{replay} case should be identical to the input of the corresponding neuron in the FB network. As the replay of spikes and the resulting postsynaptic currents and membrane potentials are not perfectly reproducible on the *Spikey* chip, the neural responses in the FB and in the FB_{replay} scenario are slightly different (compare Figures 6.5a and 6.5b). In the RAND case (Figures 6.2d and 6.5c), we use the same setup as in the FB_{replay} case. However, the spike times in each presynaptic spike train are randomized. While the average presynaptic firing rates and the shared-input structure are exactly preserved in this scenario, the spatio-temporal correlations in the presynaptic spiking activity are destroyed.

Using this setup, we first demonstrate in section 6.3.1 that active decorrelation by inhibitory feedback (Renart et al., 2010a; Tetzlaff et al., 2012a) is effective in heterogeneous networks with conductance-base synapses over a range of different network sizes. In section 6.3.2, we show that decreasing the level of heterogeneity by calibration of hardware neurons leads to an enhancement of this active decorrelation.

6.3.1 Decorrelation by inhibitory feedback

The time-averaged population activities in the FB, FB_{replay} and RAND scenarios are roughly identical (vertical histograms in Figures 6.5a–c; see also high-frequency power in figure 6.5e). In the FB and FB_{replay} scenario, fluctuations in the population-averaged activity are small (horizontal histograms in figure 6.5a and b). The removal of spatial and temporal correlations in the presynaptic spike trains in the RAND

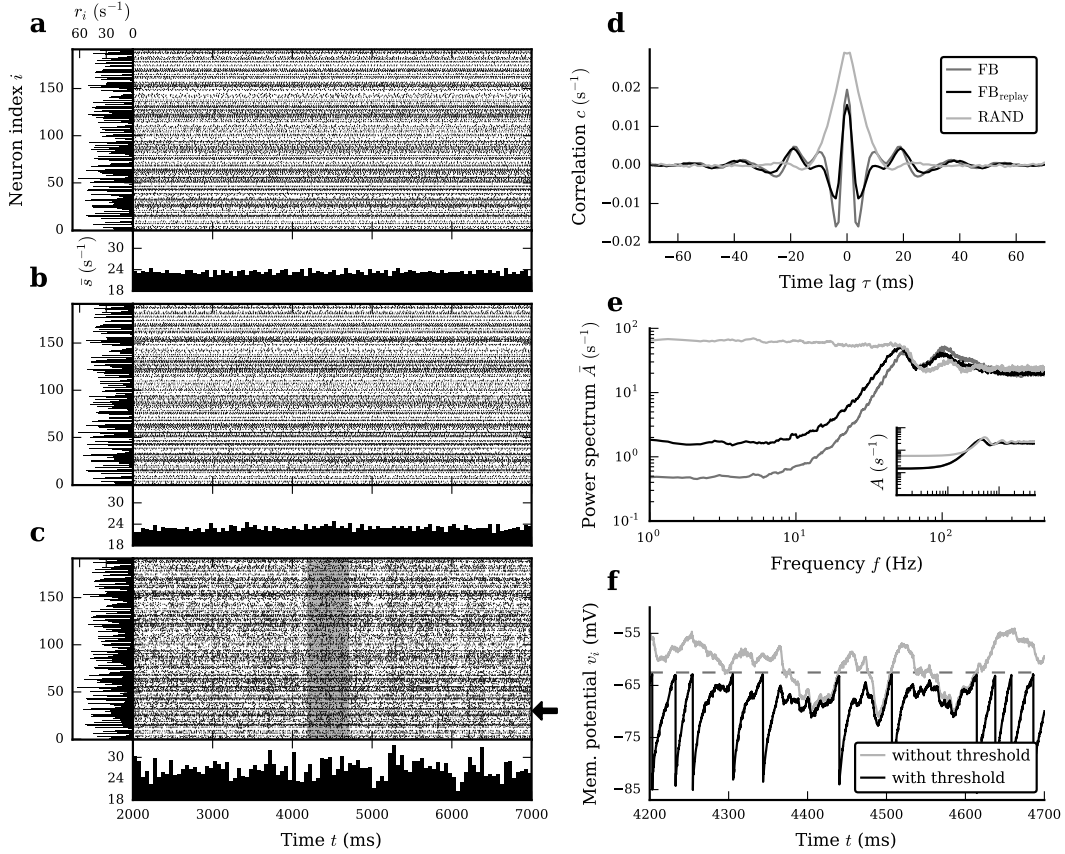


FIGURE 6.5: Typical spiking and membrane-potential activity of a random inhibitory network of LIF neurons with intact and cut feedback loop emulated on the fully calibrated system. **(a–c)** Spiking activity (raster plots), population activity $\bar{s}(t)$ (horizontal histograms; bin size 50 ms) and time-averaged single-neuron firing rates r_i (vertical histograms) in the network with intact feedback (a) and for cases where the feedback loop is cut (b and c). (a) Intact recurrent network (FB scenario). (b) Population of mutually unconnected neurons receiving input spike trains identical to those in (a) (FB_{replay} scenario). (c) As in (b), but after randomization of presynaptic spike times (RAND scenario). **(d,e)** Population-averaged cross-correlation functions $c(\tau)$ (after offset subtraction) of pairs of spike trains (d) and power spectra $\bar{A}(f)$ (e; log-log representation) of the population activity $\bar{s}(t)$ (cf. horizontal histograms in (a–c)) for the FB (dark gray), FB_{replay} (black) and RAND scenario (light gray). Inset in (e): Population-averaged power spectra $A(f)$ of individual single-cell spike trains (same scales as in main panel). Correlation functions and spectra are averaged across $M = 100$ network realizations. **(f)** Membrane potential of a neuron in the RAND scenario (with firing rate of 23.20 s^{-1} close to population average of 23.24 s^{-1} ; see black arrow in (c)) with intact (black curve) and removed threshold (gray curve; *free membrane potential*). The threshold potential is marked by the horizontal dashed line. The time frame corresponds to the gray-shaded region in (c).

case leads to a significant increase in the fluctuations of the population-averaged response activity (horizontal histogram in figure 6.5c). At low frequencies (≤ 20 Hz), the population-rate power in the FB and in the RAND case differs by about two orders of magnitudes (dark and light gray curves in figure 6.5e). This increase in low-frequency fluctuations in the RAND case is mainly caused by an increase in pairwise correlations in the spiking activity (figure 6.5d; the power spectra of individual spike trains [inset in figure 6.5e] are only marginally affected by a randomization of presynaptic spike times) (Tetzlaff et al., 2012a). In other words, shared-input correlations, i.e., those leading to large spike-train correlations in the RAND scenario, are efficiently suppressed by the feedback loop in the FB case.

On the neuromorphic hardware, the replay of network activity is not perfectly reproducible (section 6.2.4). While the across-trial variability in membrane potentials is small, postsynaptic spikes are dithered by few milliseconds (figure 6.3). In the $\text{FB}_{\text{replay}}$ case, the suppression of shared-input correlations by correlations in presynaptic spike trains is slightly less efficient as compared to the intact network (FB). The differences in the population-rate power spectra and in the spike-train correlations between the $\text{FB}_{\text{replay}}$ and RAND case, respectively, are nevertheless substantial (solid black and light gray curves in figure 6.5d and e; note the logarithmic scale; for a detailed investigation of spike dither see appendix B.6 and appendix figure B.7).

Note that the suppression of correlations and, hence, population-rate fluctuations by inhibitory feedback is restricted to low frequencies (here, to frequencies < 50 Hz; see figure 6.5e). In the remainder of this study, we will quantify pairwise correlations by the low-frequency coherence in the range 0.1–20 Hz (see section 6.2.6). At higher frequencies, the population-rate power spectra in the FB, $\text{FB}_{\text{replay}}$ and RAND case are similar. In figure 6.5e, the peaks at ~ 50 Hz and higher harmonics result from the single-cell spike-train statistics (they are also visible in the single-cell spectra; see inset): A large fraction of cells, in particular those firing at higher rates, generate regular spike trains with low inter-spike-interval (ISI) variability (cf. figure 6.6d–f). These (fast spiking) cells contribute maxima to the spike-train spectra at frequencies close to their firing rates (and higher harmonics). The structure of the population-rate spectra at higher frequencies (≥ 50 Hz) is reproduced using surrogate data where the ISI distributions of the individual neurons (and, hence, their firing rates and ISI variability) are preserved, but serial ISI correlations and cross-correlations between spike trains are destroyed (data not shown).

In the RAND case, presynaptic spike-train correlations were removed, and hence input (i.e., free-membrane-potential) correlations are exclusively determined by the number of shared presynaptic sources (equation (6.2)). If the in-degree K is fixed, input correlations will decrease with network size N (equation (6.2), light gray curve and symbols in figure 6.7a). In purely inhibitory networks with intact feedback loop (FB scenario), correlations in presynaptic spike trains are on average significantly smaller than zero (dark gray diamonds in figure 6.7b, Tetzlaff et al. (2012a)), and largely cancel the positive contribution from shared-input correlations. Average input correlations are therefore significantly reduced (black symbols in figure 6.7a). As both shared-input and spike-train correlations scale with the inverse of the network size (N^{-1} ; light gray curve in figure 6.7a and inset in figure 6.7b, respectively) (De la Rocha et al., 2007), this suppression of correlations in the FB (and $\text{FB}_{\text{replay}}$) case is observed for all investigated network sizes N . Note that output correlations are negative even though input correlations are positive. This effect is predicted by theory and also observed in linear network models as well as LIF-network simulations on conventional computers (see figure 6.9, appendix B.6, B.7 and section 6.4).

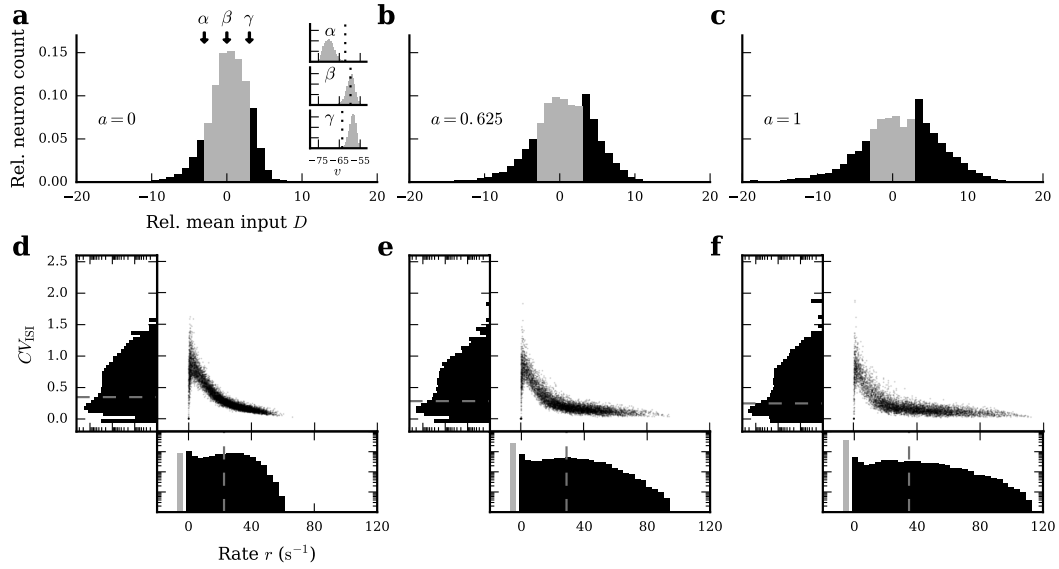


FIGURE 6.6: Modulation of network heterogeneity by leak-conductance calibration (see section 6.2.5). Input (top row) and firing statistics (bottom row) in the intact recurrent networks (FB scenarios) for fully calibrated (a and d; $a = 0$), partially calibrated (b and e; $a = 0.625$) and uncalibrated neurons (c and f; $a = 1$). **(a–c)** Effect of calibration on input statistics. Distributions of relative mean input $D = (\bar{v} - \Theta)/\sigma(v)$ (distance of time averaged free membrane potential $\bar{v}$ from firing threshold Θ in units of the standard deviation $\sigma(v)$) across the population of neurons. Gray areas in (a), (b) and (c) highlight $[-3, 3]$ intervals, containing 74%, 53% and 42% of the total mass of the distribution, respectively. Inset in (a): Distributions of free membrane potentials v for three neurons α , β and γ with $D = -3$, $D = 0$ and $D = 3$ (arrows in (a)), respectively. Dotted lines mark threshold potentials that may vary due to fixed-pattern noise. **(d–f)** Effect of calibration on spike-train statistics. Joint (scatter plots) and marginal distributions of single-neuron firing rates ν (horizontal histograms; log-linear scale) and coefficients of variation CV_{ISI} of inter-spike intervals (vertical histograms; log-linear scale). Dashed lines mark mean of firing rate (22.6 s^{-1} , 28.7 s^{-1} , 34.8 s^{-1}) and CV_{ISI} distributions (0.35 , 0.28 , 0.25), respectively. Gray bars (bottom panels) represent fractions of silent neurons. Data obtained from $M = 50$ different network realizations. Percentage of dead neurons: 8%, 26%, 37%.

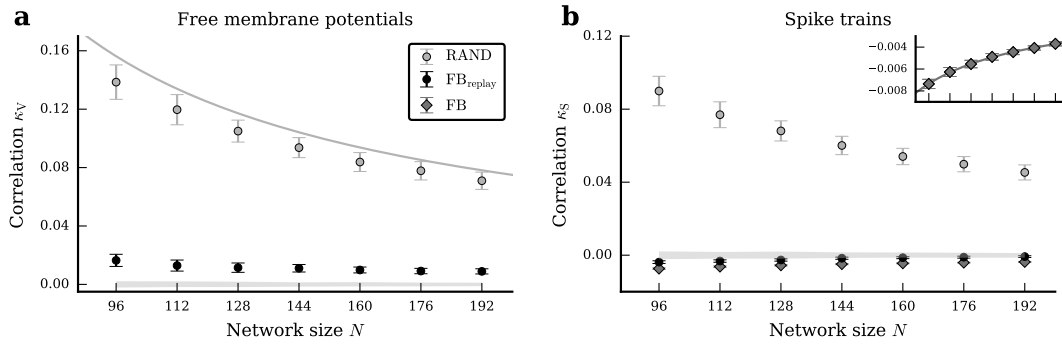


FIGURE 6.7: Dependence of population-averaged input correlations **(a)** and spike-train correlations **(b)** on the network size N , for the intact network (FB, dark gray diamonds), the FB_{replay} (black circles) and the RAND (light gray circles) case (fixed in-degree $K = 15$). Symbols and error bars denote mean and one standard deviation, respectively, across $M = 100$ network realizations (error bars are partly covered by markers). The gray curve in (a) depicts shared-input correlations in a homogeneous network (equation (6.2)). The inset in (b) shows a magnified view of the spike-train correlations in the FB case (dark gray diamonds) with a power-law fit $\sim N^{-1}$ (dark gray curve). The light gray horizontal band represents mean $\pm$ three standard deviations of (spurious) correlations in surrogate data where correlations were removed. Note that free membrane potentials cannot be recorded in the FB case (see section 6.2). Hence, there are no gray diamonds in (a).

6.3.2 Effect of heterogeneity on decorrelation

In neural networks implemented in analog neuromorphic hardware, neuron and synapse parameters vary significantly across the population of cells (fixed-pattern noise; see section 6.2.2). For a population of mutually unconnected neurons with distributed parameters, injection of a constant (supra-threshold) input current leads to a distribution of response firing rates (figure 6.4). In this study, we consider the width of this firing-rate distribution as a representation of neuron heterogeneity. It is systematically varied by calibration of leak conductances. The extent of heterogeneity is quantified by the calibration parameter a ($a = 1$ and $a = 0$ correspond to the uncalibrated and the fully calibrated system, respectively; for details, see section 6.2.5). For an unconnected population of neurons subject to constant input, the width of the firing-rate distribution increases monotonically with a .

As shown in figure 6.6, the level of heterogeneity (i.e., the calibration state a) is clearly reflected in the activity of the recurrent network (FB case). Both the width of the distribution of mean free membrane potentials (figure 6.6a–c) as well as the width of the firing-rate distribution increases with a (figure 6.6d–f; horizontal histograms). In the uncalibrated system ($a = 1$), a substantial fraction of neurons is predominantly driven by constant supra-threshold input currents and therefore generates highly regular spike trains ($CV_{\text{ISI}} \approx 0$) with high firing rates ($\nu > 60 \text{ s}^{-1}$). Simultaneously, about 37% of the neurons are silent ($\nu = 0 \text{ s}^{-1}$). Neurons with intermediate firing rates ($0 \text{ s}^{-1} < \nu < 15 \text{ s}^{-1}$), however, show quite irregular activity ($CV_{\text{ISI}} > 0.5$). After calibration, the firing-rate distribution is narrowed. For $a = 0$, the fraction of silent neurons is reduced to about 8%. Maximum rates are limited to $< 66 \text{ s}^{-1}$. Note that our calibration routine compensates only for the distribution of neuron parameters, but not for the heterogeneity in synapse properties (synaptic weights, synaptic time constants; see section 6.4). Even for the fully calibrated network ($a = 0$), the firing-rate distribution is therefore still broad. In the RAND case,

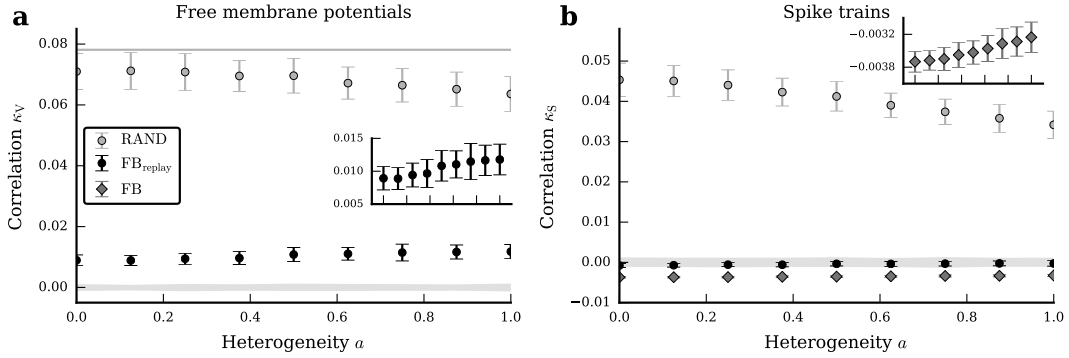


FIGURE 6.8: Dependence of population-averaged input correlations **(a)** and spike-train correlations **(b)** on the heterogeneity of the neuromorphic substrate for the intact network (FB, dark gray diamonds), the $\text{FB}_{\text{replay}}$ (black circles) and the RAND (light gray circles) case. Symbols and error bars denote mean and one standard deviation, respectively, across $M = 100$ network realizations (error bars are partly covered by markers). The gray line in (a) depicts shared-input correlations in a homogeneous network (equation (6.2)). Insets in (a) and (b) depict magnified views of input correlations in the $\text{FB}_{\text{replay}}$ case and spike-train correlations in the FB case, respectively. The light gray horizontal band represents mean $\pm$ three standard deviations of (spurious) correlations in surrogate data where correlations were removed. Note that free membrane potentials cannot be recorded in the FB case (see section 6.2). Hence, there are no gray diamonds in (a).

we obtain similar firing-rate and ISI statistics as in the FB case (appendix B.8).

For all levels of heterogeneity attainable by our calibration procedure ($a \in [0, 1]$), input and output correlations are significantly suppressed by the recurrent-network dynamics (cf. black and dark gray vs. light gray symbols in figure 6.8). In a homogeneous, random (Erdős-Rényi) network with fixed in-degree K and linear sub-threshold dynamics, the contribution of shared input to the input (free-membrane-potential) correlation is given by the network connectivity K/N (equation (6.2); Tetzlaff et al., 2012a) (thin light gray curves in figure 6.7a and figure 6.8a). Non-linearities in synaptic and/or spike-generation dynamics (De la Rocha et al., 2007) as well as heterogeneity in neuron (and synapse) parameters lead to a suppression of this contribution (equation (6.3), Padmanabhan and Urban (2010)). Here, we refer to this type of decorrelation as *feedforward decorrelation*. In fact, in our setup the input and spike-train correlations in the RAND case decrease with increasing heterogeneity (light gray symbols in figure 6.8). Even in the fully calibrated case input correlations are slightly smaller than K/N (gray symbols vs. thin gray curve in figure 6.8a) due to remaining heterogeneities. Spike-train correlations decrease slightly faster with increasing heterogeneity than input correlations. These observations indicate that both the synaptic integration and spike generation are affected by heterogeneities on hardware. To illustrate the different effects of heterogeneity in synaptic integration and spike generation, we performed network simulations on conventional computers where we distributed either firing thresholds (figure 6.9) or synaptic weights (appendix figure B.4). In the RAND case, the decrease of input correlations on the hardware can be attributed to an increase in heterogeneity in parameters affecting synaptic integration (compare light gray symbols in figure 6.8a and appendix figure B.4a). Heterogeneity in spike thresholds, in contrast, does not affect input correlations in the RAND scenario, but strongly reduces spike-train correlations (light gray symbols in figure 6.9). Overall, feedforward decorrelation, i.e. the suppression of correlations in the RAND case, becomes more effective in networks

with heterogeneous cell parameters.

Despite this enhancement of feedforward decorrelation, input and output correlations increase with the level of heterogeneity in the presence of an intact feedback loop (FB and FB_{replay} scenarios; black and dark gray symbols in figure 6.8). We attribute this effect to a weakening of the effective feedback loop in the recurrent circuit: In heterogeneous networks with broad firing-rate distributions, neurons firing with low or high rates, corresponding to mean inputs far below or far above firing threshold (see figure 6.6a–c), are less sensitive to input fluctuations than moderately active neurons (see appendix figure B.2). Hence, they contribute less to the overall feedback. In consequence, feedback decorrelation is impaired by heterogeneity (see also section 6.4).

6.4 Discussion

We have shown that inhibitory feedback effectively suppresses correlations in heterogeneous recurrent neural networks of leaky integrate-and-fire (LIF) neurons with nonlinear subthreshold dynamics, emulated on analog neuromorphic hardware (*Spikey*; Schemmel et al. (2006) and Pfeil et al. (2013b)). Both input and output correlations are substantially smaller in networks with intact feedback loop (FB), as compared to the case where the feedback is replaced by randomized input while preserving the connectivity structure and presynaptic firing rates (RAND). Our results hence show that active decorrelation of network activity by inhibitory feedback (Renart et al., 2010a; Tetzlaff et al., 2012a) is a general phenomenon which can be observed in realistic, highly heterogeneous networks with nonlinear interaction and sufficiently strong negative feedback. Moreover, the study serves as a proof-of-principle that network activity can be efficiently decorrelated even on heterogeneous hardware, which can be exploited in functional applications, e.g., in the neuromorphic algorithms developed by Pfeil et al. (2013b) and Schmuker, Pfeil, and Nawrot (2014).

Partial calibration of hardware neurons allowed us to modulate the level of network heterogeneity and, therefore, to systematically study its effect on correlations in the network activity. The analysis revealed two counteracting contributions: As shown in previous studies (e.g., Padmanabhan and Urban, 2010), neuron heterogeneity decorrelates (shared) feedforward input (feedforward decorrelation). On the other hand, however, heterogeneity impairs feedback decorrelation (see next paragraph). In our network model, this weakening of feedback decorrelation is the dominating factor. Overall, we observed a slight increase in correlations with increasing level of heterogeneity. We cannot exclude that feedforward decorrelation may play a more significant role for different network configurations (e.g., different connection strengths or network topologies, different structure of external inputs, different types of heterogeneity). Our study demonstrates, however, that heterogeneity is not necessarily suppressing correlations in recurrent systems. In this context, it would be interesting to investigate the interplay of signal and noise correlations in the presence of network heterogeneities in recurrent systems. We leave this intriguing topic to future studies.

As shown in Tetzlaff et al. (2012a), feedback decorrelation in recurrent networks becomes more (less) efficient with increasing (decreasing) strength of the *effective* negative feedback. For networks of spiking neurons, the effective connection strength w_{ij} between two neurons j and i corresponds to the total number

of extra spikes emitted by neuron i in response to an additional input spike generated by neuron j (see, e.g., London et al. (2010)). Assuming that the effect of a single additional input spike is small, the effective connectivity can be obtained by linear-response theory (De la Rocha et al., 2007). Note that the effective weights w_{ij} depend on the working point, i.e., the average firing rates of all pre- and postsynaptic neurons (mathematically, w_{ij} is given by the derivative of the stationary response firing rate $r_i = \phi_i(r_1, \dots, r_j, \dots, r_N)$ of neuron i with respect to the input firing rate r_j , evaluated at the working point; for details, see Tetzlaff et al. (2012a)). Neurons firing at very low or very high rates are typically less sensitive to input fluctuations than neurons firing at intermediate rates (due to the shape of the response function $\phi_i(r_1, \dots, r_N)$). Their dynamical range is reduced. In consequence, they hardly mediate feedback in a recurrent network. In heterogeneous networks with broad distributions of firing rates, the number of these insensitive neurons is increased. Hence, the effective feedback is weakened (see appendix B.5). We can qualitatively reproduce this effect of heterogeneity on correlations in recurrent networks (FB case) by means of a simplified linear rate model where increasing heterogeneity is described as a decrease in the effective-weight amplitudes (see appendix B.7). A more quantitative analysis requires an explicit mapping of the synaptic weights in the LIF-neuron network to the effective weights of the linear model (as in, e.g., (Grytskyy et al., 2013a)) in the presence of distributed firing rates. We commit this task to future studies. Note that the rate dependence of the effective weights and the resulting effects on correlations are consistent with our observation that neuron pairs with very low firing rates exhibit spike-train correlations close to zero, whereas pairs with high firing rates are positively correlated (see appendix B.9). Pairs with one neuron firing at an intermediate rate often exhibit negative spike-train correlations. As shown in Renart et al. (2010a) and Tetzlaff et al. (2012a), these negative spike-train correlations are essential for compensating the positive contribution of shared inputs to the total input correlation (at least in purely inhibitory networks). Narrowing the firing rate distribution (e.g., by calibration of hardware neurons) increases the number of neurons contributing to the negative feedback, which, in turn, leads to more neuron pairs with negative spike-train correlations and, therefore, to smaller overall correlations.

Seemingly contrary to our findings, Bernacchia and Wang (2013) report a decrease in correlations with increasing level of heterogeneity. The results of their study are obtained for a linear network model, which can be considered the outcome of the linearization procedure described above. Hence, the connectivity of their model corresponds to an effective connectivity (see above). Their study neglects the rate (working-point) dependence of the effective weights and can therefore not account for the effect of firing-rate heterogeneity. In Bernacchia and Wang (2013), heterogeneity is quantified by the variance of the (effective) weight matrix (Equations 2.2 and 2.4 in Bernacchia and Wang (2013)). For sparse connectivity matrices (with a large number of zero elements), the variance of the weight matrix reflects not only the width of the non-zero-weight distribution, but also its mean (Equation 2.4 in Bernacchia and Wang (2013)). For networks of nonlinear spiking neurons, heterogeneities in neuron and/or synapse parameters broadens the distribution of non-zero effective weights, but may simultaneously reduce its mean (see above, appendix B.5, and Roxin et al. (2011) and Grytskyy et al. (2013a)). Hence, the variance of the full weight matrix may decrease (for illustration, see appendix figure B.9). In other words, increasing heterogeneity in the nonlinear system may correspond to decreasing heterogeneity in the linearized system. A direct test of this hypothesis requires an explicit linearization of the nonlinear heterogeneous system.

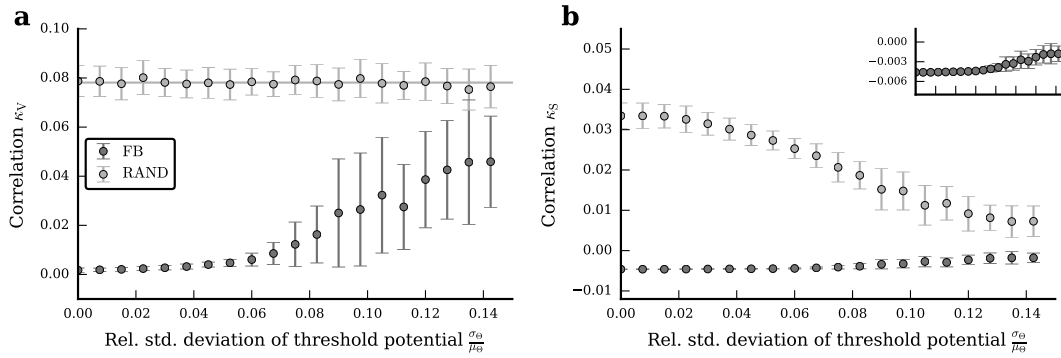


FIGURE 6.9: Dependence of population-averaged input correlations **(a)**, and spike-train correlations **(b)** on the width of threshold distributions in networks simulated with NEST (Gewaltig and Diesmann, 2007) and PyNN (Davison et al., 2008b), for the intact network (FB, dark gray circles) and the RAND (light gray circles) case. Symbols and error bars denote mean and standard deviation, respectively, across $M = 30$ network realizations (error bars are partly covered by markers). The gray line in (a) depicts shared-input correlations in a homogeneous network (equation (6.2)). The inset in (b) shows a magnified view of the spike-train correlations in the FB case. Note that in simulations the $\text{FB}_{\text{replay}}$ is identical to the FB case, and is hence not shown. For details see appendix B.6.

The results of this study were obtained by network emulations on analog neuromorphic hardware. We reproduced the main findings by means of conventional computer simulations of LIF-neuron networks with distributed firing thresholds (see figure 6.9). The focus on threshold heterogeneity allows us to isolate the effect of firing-rate distributions on correlations. It does not affect shared-input correlations (see section 6.2.6). Although networks simulated on conventional computers and those emulated on the neuromorphic hardware differ in several respects (e.g., in the exact implementation of heterogeneity or the synapse model; compare appendix table B.5 and B.6 to table B.1 and B.2, respectively), the qualitative results are very similar: In networks with intact feedback loop, input and output correlations are substantially reduced (as compared to the case where the feedback is replaced by randomized input), but increase with the extent of heterogeneity. As predicted by the theory for homogeneous inhibitory networks, we observe positive input correlations and negative output correlations (see Equation 21 in Tetzlaff et al. (2012a) and in the paragraph which follows; see also Helias, Tetzlaff, and Diesmann (2013) and appendix B.7). Further, note that heterogeneity in neuron parameters does not “average out” in larger networks. Upscaling the network size by a factor of 25 ($N = 4800$, in-degree $K = 375$) yields smaller spike-train correlations, but the qualitative results are similar to those obtained for the smaller network ($N = 192$, $K = 15$) emulated on the *Spikey* chip (compare figure 6.8 to appendix figure B.3).

In networks with intact feedback loop (FB and $\text{FB}_{\text{replay}}$ scenarios), the precise spatio-temporal structure of spike trains arranges such that the self-consistent input and output correlations are suppressed. Perturbations of this structure in the local input typically lead to an increase in correlations (Tetzlaff et al., 2012a). In this study, we demonstrate this by replaying spiking activity after randomization of spike times, i.e., by replacing the time of each input spike by a random number uniformly drawn from the full emulation time interval $[0, T)$ (RAND case). However, even subtle modifications of input spike trains, such as random dither of spike times by few milliseconds, lead to an increase of correlations. On the neuromorphic hardware, replay of spike trains is not entirely reproducible (see section 6.2.4

and appendix B.11). Hence, spike-train correlations measured in the $\text{FB}_{\text{replay}}$ mode are slightly larger than in the FB case. We would expect the same effect on the input side (free membrane potentials). Due to hardware limitations, however, we can measure input correlations only in replay mode ($\text{FB}_{\text{replay}}$ or RAND), but not in the fully connected network (FB). Therefore, all reported input correlations are likely to be slightly overestimated. In conventional network simulations, we mimicked the effect of unreliable replay by input-spike dithering and, indeed, find a gradual increase in input and output correlations (see appendix figure B.6). These results seem to be contrary to the study by Rosenbaum and Josic (2011), in which synaptic noise leads to a decrease of output correlations in a feedforward scenario. In our case, spike-train correlations, which suppress shared-input correlations, are removed by dithering spikes, thereby increasing correlations on the output side. In Rosenbaum and Josic (2011), in contrast, spike-train correlations are always zero, and shared-input correlations are decreased by synaptic failure, explaining the decreased output correlations. We attribute this contradiction to the missing feedback loop in their system, and expect correlations to increase in recurrent networks subject to similar perturbations.

Despite the imperfect replay of input spikes, the decorrelation effect is clearly visible in hardware emulations, both on the input and on the output side. The reproducibility of emulations on neuromorphic hardware could be improved by stabilizing the environment of the system, e.g., the chip temperature or the support electronics (under development). Analog hardware, however, will never reach the level of reproducibility of digital computers. But note that, similar to analog hardware, biological neurons exhibit a considerable amount of trial-to-trial variability, even under controlled in-vitro conditions (Mainen and Sejnowski, 1995b). So far, the details of how neuronal noise, for example, stochastic synapses (spontaneous post-synaptic events, stochastic spike transmission, synaptic failure (Ribault, Sekimoto, and Triller, 2011b)), affects correlations in recurrent neural circuits remain unclear.

Although different *Spikey* chips exhibit different realizations of fixed-pattern noise, they show a comparable extent of heterogeneity and yield results which are qualitatively similar to those presented in this chapter (appendix B.10).

We have shown that negative feedback in recurrent circuits can efficiently suppress correlations, even in highly heterogeneous systems such as the analog neuromorphic architecture *Spikey*. Correlations can be further reduced by minimizing the level of network heterogeneity. In this study, we reduced the level of heterogeneity through calibration of neuron parameters in the unconnected case (see section 6.2.5). The calibration could, in principle, be improved by calibrating neuron (and possibly synapse) parameters in the full recurrent network. Such calibration procedures are however time consuming and cumbersome. In biological substrates, homeostasis mechanisms (Marder and Goaillard, 2006; O’Leary et al., 2014) keep neurons in a responsive regime and reduce the level of firing-rate heterogeneity in a self-regulating manner. Future neuromorphic devices could mimic this behavior, thereby reducing the necessity of time consuming calibration procedures. Alternatively, the analog circuits could be optimized to reduce fixed-pattern noise. This would likely require the allocation of more chip resources, hence reducing the network size per chip area.

For simplicity, this work focuses on purely inhibitory networks (as in Mar et al. (1999)). This demonstrates that decorrelation by inhibitory feedback does not rely on a dynamical balance between excitation and inhibition (note that the external “excitatory” drive is constant in our model) (Tetzlaff et al., 2012a; Helias, Tetzlaff, and Diesmann, 2014a). Previous studies have shown that, for the homogeneous case, decorrelation by inhibitory feedback is a general phenomenon, which also occurs in

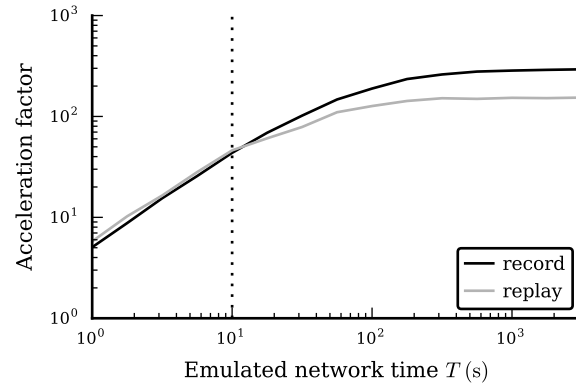


FIGURE 6.10: Acceleration factor as a function of emulated network time T for the record (black) and the replay case (gray). The acceleration factor is defined as the ratio between the emulated network time T (in biological time) and the execution time (wall clock time). In the record case, a network realization is generated on the host computer and uploaded to the chip. During the subsequent emulation, spike trains are recorded. In the replay case, spikes are replayed and the membrane potential of one neuron is recorded with full sampling frequency (9.6 kHz). The execution time covers the full data flow from a network description in `PyNN` to the emulation on the *Spikey* system and back to the network representation in `PyNN`. The time-averaged population firing rate is $\bar{\nu} = 23.7(2) \text{ s}^{-1}$. The vertical dashed line depicts the runtime used in this study. The hardware system has to be initialized once before usage ($< 1 \text{ s}$), which is not considered here.

excitatory-inhibitory networks, provided the overall inhibition is sufficiently strong (which is typically the case to ensure stability) (Renart et al., 2010a; Tetzlaff et al., 2012a; Bernacchia and Wang, 2013; Helias, Tetzlaff, and Diesmann, 2014a). For the heterogeneous case, computer simulations of excitatory-inhibitory networks show qualitatively the same results as purely inhibitory networks (compare figure 6.8 to appendix figure B.5), confirming that our results generalize to the case of mixed excitatory-inhibitory coupling.

Similar to our study, Giulioni et al. (2012) use a theory-guided approach to implement, verify and investigate network dynamics on analog neuromorphic hardware. In their study, an attractor network is implemented that is inspired by a mean-field model. Due to heterogeneities in synaptic efficacies on the hardware, stability analysis of attractor states requires the authors to measure *effective response functions* of populations of hardware neurons. To this end, they replace recurrent connections in one population of neurons by external input. This allows them to measure the firing rate of the population as a function of the external input, while the activity of the population is in equilibrium with that of other recurrently connected populations. This study represents another example illustrating that investigations of actual hardware emulations are a prerequisite for successful application of analog neuromorphic hardware.

This study demonstrates that the *Spikey* system has matured to a level that permits its use as a tool for neuroscientific research. For the results presented in this study, we recorded in total 10^{11} membrane-potential and spike-train samples, representing more than 100 days of biological time. Due to the 10^4 -fold acceleration of the *Spikey* chip, this corresponds to less than 15 minutes in the hardware-time domain. Interfacing the hardware system, however, reduces the acceleration to an approximately 50-fold speed-up (figure 6.10). The translation between the network description and its hardware representation claims the majority of execution time,

more than the network emulation and the transfer of data to and from the hardware system together. Encoding and decoding spike times on the host computer is particularly expensive. Obviously, the system could be optimized by processing the data directly on the hardware, or by choosing a data representation which is closer to the format used on the *Spikey* chip, but this would impair user-friendliness, and hence, the effectiveness of prototyping. While the *Spikey* system permits the monitoring of the spiking activity of all neurons simultaneously, access to the membrane potentials is limited to a single (albeit arbitrary) neuron in each emulation run. Monitoring of membrane potentials of a population of n neurons therefore requires n repetitions of the same emulation. Extending the hardware system to enable access to the membrane potentials of at least two neurons simultaneously would allow for a direct observation of input correlations in the intact network (and thereby avoid problems with replay reproducibility; see above) and reduce execution time (the *Spikey* chip itself permits recording of up to eight neurons in parallel, the support electronics, however, does not). While the *Spikey* system does not significantly outperform conventional computers in terms of computational power, emulations on this system are much more energy efficient (appendix B.3). A substantial increase of computational power is expected for large systems exploiting the scalability of this technology without slow-down (Schemmel et al., 2010a).

The results from this chapter demonstrate that recurrent networks can be employed as noise sources for physical implementations of approximate inference as proposed in chapter 5 since shared-input correlations are efficiently suppressed also in strongly heterogeneous neuromorphic substrates. A full neuromorphic-hardware implementation of the framework is, however, still pending. In the next chapter we return to the domain of numerical simulation to investigate alternative sources of noise, including stochasticity in the synaptic transmission, and explore the possibility of probabilistic computations in sampling networks of non-Boltzmann type.

Chapter 7

Approximate inference in non-Boltzmann networks

Boltzmann machines (section 2.5, Hinton, Sejnowski, and Ackley (1984b)) have been proposed as models for approximate inference in biological neural networks (Buesing et al., 2011). Owing to their symmetric connections and the particular form of the activation function of the individual units, they exhibit equilibrium dynamics. It is hence possible to assign an energy to each state of the system, similar to physical systems in equilibrium statistical mechanics. However, Boltzmann machines rely on a particular type of stochasticity and symmetric synaptic connections, conditions difficult to meet in biological systems. In this chapter, we explore computational properties of sampling networks derived from Boltzmann machines that employ different activation functions for the individual units, implement noisy synaptic weights or possess asymmetric connectivity. One of the key properties of Boltzmann machines is their ability to perform inference on the joint distribution embodied by the network. We now discuss briefly how one can perform fast inference in Boltzmann machines by constraining their sampling dynamics.

7.1 Fast inference via constraints in Boltzmann machines

We consider a Boltzmann machine with stationary distribution $p(\mathbf{s})$ over a set of n binary random variables (RVs). To perform inference, we need to compute the conditional distribution $p(\mathbf{s}_k | \mathbf{s}_{\setminus k})$ over k RVs after observing the states of the remaining $n - k$ RVs. Without loss of generality we assume the RVs to be ordered such that the first k entries in $\mathbf{s}$ correspond to unobserved, denoted by $\mathbf{s}_k$, and the remaining $n - k$ to observed RVs, denoted by $\mathbf{s}_{\setminus k}$. In the following, the observed states are also collectively referred to as “evidence”. Computing the conditional distribution over a subset of k RVs given the evidence involves a sum over all possible states of the unconditioned units:

$$p(\mathbf{s}_k | \mathbf{s}_{\setminus k}) = \frac{p(\mathbf{s})}{\sum_{\{\mathbf{s}_k\}} p(\mathbf{s})} \quad (7.1)$$

If k becomes large, evaluating the denominator becomes infeasible and one would hence employ a sample-based approach (cf. section 1.2). A Boltzmann machine provides samples from the joint distribution $p(\mathbf{s})$ over all RVs. To sample from the conditional, one could collect samples from the marginal distribution over $\mathbf{s}_k$ (equation (2.17)), only counting samples if the remaining units are in the state compatible with the current evidence. While this approach allows one to approximate the exact conditional given specific evidence $\mathbf{s}_{\setminus k}$, one discards most samples if the number of

constrained variables is large.¹ Fortunately, the dynamics of a Boltzmann machine can generate samples directly from the conditional distribution $p(\mathbf{s}_k | \mathbf{s}_{\setminus k})$ by appropriately constraining the sampling dynamics, as we discuss in the following.

By suppressing states in which a unit is on that should be off, and vice versa, any distribution $p(\mathbf{s})$ over binary RVs can be transformed into a new distribution in which states that conflict with the given evidence are exponentially unlikely:

$$\tilde{p}(\mathbf{s}) \sim p(\mathbf{s}) e^{\alpha \tilde{\mathbf{b}}^T \mathbf{s}}, \quad (7.2)$$

with $\alpha \in \mathbb{R}^+$. Here $\tilde{\mathbf{b}}$ is defined as follows:

$$\tilde{b}_i := \begin{cases} 1 & \text{if } i \geq k \text{ and } s_i = 1 \text{ observed} \\ -1 & \text{if } i \geq k \text{ and } s_i = 0 \text{ observed} \\ 0 & \text{else} \end{cases} \quad (7.3)$$

For any α , the conditional of the new distribution $\tilde{p}(\mathbf{s}_k | \mathbf{s}_{\setminus k})$ is identical to the conditional $p(\mathbf{s}_k | \mathbf{s}_{\setminus k})$ (see also Biermann, 2016):

$$\begin{aligned} \tilde{p}(\mathbf{s}_k | \mathbf{s}_{\setminus k}) &= \frac{\tilde{p}(\mathbf{s})}{\tilde{p}(\mathbf{s}_{\setminus k})} \\ &= \frac{p(\mathbf{s}) e^{\alpha \tilde{\mathbf{b}}^T \mathbf{s}}}{\sum_{\{\mathbf{s}_k\}} p(\mathbf{s}) e^{\alpha \tilde{\mathbf{b}}^T \mathbf{s}}} \\ &= p(\mathbf{s}_k | \mathbf{s}_{\setminus k}). \end{aligned} \quad (7.4)$$

Here the exponential cancels, since $\tilde{b}_i$ is by definition zero for all $i < k$ and $e^{\alpha \tilde{\mathbf{b}}^T \mathbf{s}}$ hence independent of the sum. To perform fast inference, we would like to sample directly from the conditional $p(\mathbf{s}_k | \mathbf{s}_{\setminus k})$. As we show in the following, in Boltzmann machines the marginal of the modified distribution over the unconditioned variables for large α is identical to the conditional of the original distribution:

$$\lim_{\alpha \rightarrow \infty} \tilde{p}(\mathbf{s}_k) \equiv p(\mathbf{s}_k | \mathbf{s}_{\setminus k}). \quad (7.5)$$

The marginal is given by:

$$\tilde{p}(\mathbf{s}_k) = \int d\mathbf{s}_{\setminus k} \tilde{p}(\mathbf{s}_k | \mathbf{s}_{\setminus k}) \tilde{p}(\mathbf{s}_{\setminus k}).$$

By our definition of $\tilde{p}$ (equation (7.2)), for large α the second term will only have non-negligible entries for those states which do not violate the evidence. For example, a unit i in the on state ($s_i = 1$) with contrary evidence ($\tilde{b}_i = -1$), will contribute $e^{-\alpha}$. For $\alpha \rightarrow \infty$ the second term hence reduces to a product of delta functions, picking out those states which are compatible with the evidence. We hence obtain

$$\lim_{\alpha \rightarrow \infty} \tilde{p}(\mathbf{s}_k) \equiv \tilde{p}(\mathbf{s}_k | \mathbf{s}_{\setminus k}), \quad (7.6)$$

which according to equation (7.4) is identical to the unmodified conditional $p(\mathbf{s}_k | \mathbf{s}_{\setminus k})$. In general it is unclear how to implement the constraints on states conflicting with the evidence (equation (7.2)) in the sampling dynamics. For Boltzmann distributions (equation (2.13)) the constraints are easy to implement, since we can

¹If all states would be equally likely, one would only count every 2^{n-k} th sample.

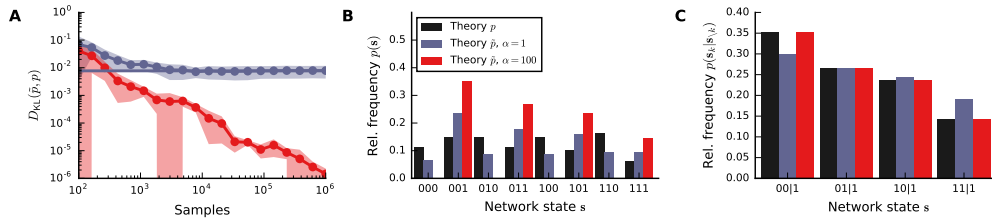


FIGURE 7.1: **Fast inference via constraints in Boltzmann machines.** **A:** Sampling error $D_{\text{KL}}(\tilde{p}, p)$ between $\tilde{p}(s_k)$ with biases modified according to equation (7.7) and $p(s_k|s_{\setminus k})$ computed via equation (7.1) as a function of the number of collected samples. Error bands represent mean $\pm$ SEM over 5 random network realizations. Solid line indicates D_{KL} for infinite sampling duration (only visible for $\alpha = 1$). **B:** Joint distribution $p(s)$ (black) and constrained joint distributions $\tilde{p}(s)$ with $\alpha = 1$ (dark purple) and $\alpha = 100$ (red), respectively. **C:** Conditional distribution $p(s_k|s_{\setminus k})$ and marginal of constrained distributions $\tilde{p}(s_k)$ for $\alpha \in \{1, 100\}$.

write the modified distribution $\tilde{p}$ as the original distribution with a change in biases

$$\begin{aligned}\tilde{p}(s) &= \frac{1}{\tilde{Z}} p(s) e^{\alpha \tilde{\mathbf{b}}^T s} \\ &= \frac{1}{\tilde{Z}} \exp\left(\frac{1}{2} s^T W s + (\mathbf{b}^T + \alpha \tilde{\mathbf{b}}^T) s\right).\end{aligned}$$

All units in $s_{\setminus k}$ that should be in the on state hence receive an additional positive bias, while all units that should be in the off state receive a negative bias (cf. equation (7.3)):

$$\mathbf{b} \rightarrow \mathbf{b} + \alpha \tilde{\mathbf{b}}. \quad (7.7)$$

We will henceforth refer to shifts in biases to implement evidence (equation (7.7)) as “constraining” and to the distributions $\tilde{p}$ accordingly as the “constrained distribution”. In conclusion, Boltzmann machines can sample from the exact conditional distributions by constraining the sampling dynamics according to the given evidence, as long as α is large enough (figure 7.1, see also Welling and Teh (2003)). This ability to perform fast inference is one of the key properties of Boltzmann machines. A priori it is not clear whether a similar procedure also allows us to perform fast inference in non-Boltzmann networks, as equation (7.7) depends on the specific form of p .

7.2 Computing the stationary distribution of arbitrary sampling networks

The logistic activation function of the stochastic units in Boltzmann machines can be interpreted as deterministic units with a Heaviside activation function receiving external noise with a particular distribution (section 5.2.3, Coolen (2001)). A more natural candidate for the additive noise would have Gaussian distribution, for example arising from a large number of uncorrelated inputs (cf. chapter 5, Hinton, Sejnowski, and Ackley (1984b)). While a closed-form expression for the stationary distribution of network states is not known for this case, we can use an alternative approach to compute the exact distribution of network states by taking into account all possible transitions between states.

We consider a network of binary units with network states $\mathbf{s}$, weights w_{ij} , biases b_i , with no self connections $w_{ii} = 0$, and possibly asymmetric weights – we do not require $w_{ij} \neq w_{ji}$. In the following we assume that we update a single randomly selected unit in every step, equivalent to a time driven update at Poissonian time points (section 5.2.1). To update the network state, we choose unit i , compute its total input field h_i and set it to the active state with a probability determined by its activation function F_i . This defines a discrete-state, discrete-time stochastic process – a Markov chain on the binary vectors $\mathbf{s}$. In the following we review a few key properties of Markov systems that will lead to an expression for the stationary state distribution in networks with arbitrary synaptic weights and activation functions. For additional details we refer to the extensive literature on the subject (e.g., Gardiner, 1985; Van Kampen, 1992; Kelly, 2011).

Recall that for a Markov chain the probability of the next state is independent of all other previous states, if the previous state is known:

$$p(\mathbf{s}_t | \mathbf{s}_{t-1}, \dots, \mathbf{s}_{t-k}, \dots) \equiv p(\mathbf{s}_t | \mathbf{s}_{t-1}). \quad (7.8)$$

Using the product and sum rules (Jaynes, 2003b) we can write the full distribution of network states at time t as a function of this conditional probability and the distribution at $t - 1$:

$$p(\mathbf{s}_t) = \sum_{\{\mathbf{s}_{t-1}\}} p(\mathbf{s}_t | \mathbf{s}_{t-1}) p(\mathbf{s}_{t-1}). \quad (7.9)$$

Consequently $p(\mathbf{s}_t | \mathbf{s}_{t-1})$, the probability to transition from state $\mathbf{s}_{t-1}$ to states $\mathbf{s}_t$, is referred to as the transition operator T of the Markov chain, $T(\mathbf{s}_t, \mathbf{s}_{t-1}) := p(\mathbf{s}_t | \mathbf{s}_{t-1})$. Since $p(\mathbf{s}_t | \mathbf{s}_{t-1})$ is a probability distribution, it must be normalized:

$$\sum_{\{\mathbf{s}_t\}} T(\mathbf{s}_t, \mathbf{s}_{t-1}) = 1, \quad (7.10)$$

which is equivalent to the requirement that probability mass must not disappear through the dynamics of the chain. In the case of countable, discrete state spaces and discrete time, equation (7.9) can be interpreted as a matrix-vector product that propagates a vector of probabilities from one time step to the next. In this context T is called the transition matrix, and we can write equation (7.9) as

$$\mathbf{p}_t = T \mathbf{p}_{t-1}. \quad (7.11)$$

Here T_{ij} denotes the probability to transition from state j at time step $t - 1$ to state i at time step t . By choosing a particular activation function F_i for the binary units, we implicitly define the transition matrix T (see below). For a large number of state transitions, the transition operator can evolve an arbitrary initial distribution $\mathbf{p}_0$ into a distribution $\mathbf{p}^*$ which is invariant under additional applications of the operator:

$$\mathbf{p}^* = T \mathbf{p}^*. \quad (7.12)$$

This distribution is referred to as the stationary distribution of the Markov chain and is guaranteed to exist if the Markov chain has the following properties:

- time homogeneity: the transition operator does not depend on the absolute time $T(\mathbf{s}_t, \mathbf{s}_{t-1}) = T(\mathbf{s}_{t+k}, \mathbf{s}_{t+k-1})$

- irreducibility: every state can be reached from every other state in a finite number of steps
- aperiodicity: none of the states are revisited with probability 1 after a fixed number of steps k

In view of equation (7.12), the stationary distribution $\mathbf{p}^*$ is an eigenvector for the transition matrix T with eigenvalue 1. Since T is column stochastic ($\sum_i T_{ij} = 1$, equation (7.10)) a left-handed eigenvector with eigenvalue 1 exists, the vector $\mathbf{1} = (1, \dots, 1)^T$. Since T is a square, real-valued matrix, it must also have a right-handed eigenvector with the same eigenvalue (Gantmacher, 1990; Kelly, 2011). One can hence determine the stationary distribution $\mathbf{p}^*$ by finding the right-handed eigenvector of T of the eigenvalue 1.

A sufficient, but not necessary, condition for the existence of a stationary distribution is the so-called detailed-balance condition:

$$T_{ij}p_j^* = T_{ji}p_i^*. \quad (7.13)$$

Detailed balance implies that the probability fluxes from state i to state j , and vice versa, exactly balance in the stationary state of the Markov chain, leading to time-reversible dynamics. This time-reversal symmetry excludes the possibility of representing spatio-temporal patterns in Boltzmann machines. For systems with different activation functions or non-symmetric connectivity, the detailed-balance condition is generally not fulfilled (see below), leading to a nonzero probability flux also after initial transients, with $T_{ij}p_j^* - T_{ji}p_i^* \neq 0$ in general. A non-vanishing net probability flux would allow systems to exhibit more interesting dynamics including the representation of particular temporal sequences of network states, likely to increase their computational power (Apolloni and Falco, 1991; Li and Dayan, 1999).

For the propagation of network states considered here, which is similar to Gibbs sampling as only a single unit is updated at a time, T is a sparse matrix: Besides remaining in the current network state, only transitions between network states that differ in the state of exactly one unit are allowed. To find the stationary distribution $\mathbf{p}^*$, we construct T from all possible transitions between states. In the following we use $\mathbf{s}^{i+}$ to denote the state $\mathbf{s}$ in which the i th unit is on, and correspondingly use $\mathbf{s}^{i-}$ if the i th unit is in the down state:

$$\mathbf{s}^{i+} := (s_0, \dots, \underbrace{1}_{i\text{th position}}, \dots, s_n)^T.$$

The off-diagonal entries of the transition matrix are proportional to the activation functions F_i of the individual units (note that i and j label network states whereas k labels individual units):

$$T(\mathbf{s}_j, \mathbf{s}_i) \sim \begin{cases} F(h_k(\mathbf{s}_i)) & \text{for } \mathbf{s}_i = \mathbf{s}^{k-}, \mathbf{s}_j = \mathbf{s}^{k+} \\ 1 - F(h_k(\mathbf{s}_i)) & \text{for } \mathbf{s}_i = \mathbf{s}^{k+}, \mathbf{s}_j = \mathbf{s}^{k-} \\ 0 & \text{else} \end{cases} \quad (7.14)$$

Since we exclude self-connections, the activation function of a unit is independent of the unit's state, and hence $F(h_k(\mathbf{s}^{k+})) \equiv F(h_k(\mathbf{s}^{k-}))$. The diagonal elements of T contain sums over all probabilities that leave the current state unchanged:

$$T(\mathbf{s}_i, \mathbf{s}_i) \sim \sum_k (F(h_k(\mathbf{s}_i))\delta_{k+} + (1 - F(h_k(\mathbf{s}_i)))(1 - \delta_{k+})), \quad (7.15)$$

where $\delta_{k+} := 1$ if the k th unit is in the on state and 0 otherwise. A straightforward implementation to construct T would include two loops, each over all possible states. However, since T is sparse one can significantly reduce the number of iterations in the inner loop by only considering pairs of states that differ in exactly one unit to the state of the outer loop, while also updating the corresponding diagonal elements (algorithm 2).² After the matrix has been constructed, we need to normalize

Algorithm 2 Construction of transition matrix T for Gibbs sampling for n binary units with activation functions F_i . $0_{2^n, 2^n}$ denotes the 2^n dimensional zero matrix, $\mathbf{s}(l)$ denotes the binary vector $\mathbf{s}$ obtained from the index l by converting it into binary representation.

```

 $T \leftarrow 0_{2^n, 2^n}$ 
for all  $l < 2^n$  do
  for all  $i < n$  do
     $m \leftarrow l \text{ xor } 2^i$ 
    if  $\mathbf{s}(l)_i = 1$  then
       $T_{lm} \leftarrow T_{lm} + F_i(\mathbf{s}(m))$ 
       $T_{ll} \leftarrow T_{ll} + F_i(\mathbf{s}(l))$ 
    else
       $T_{lm} \leftarrow T_{lm} + (1 - F_i(\mathbf{s}(m)))$ 
       $T_{ll} \leftarrow T_{ll} + (1 - F_i(\mathbf{s}(l)))$ 
    end if
  end for
end for
for all columns in  $T$  do
  normalize column to one
end for

```

its columns to sum to 1 to obtain the transition matrix T . The stationary distribution is determined by finding the eigenvector to the eigenvalue 1 (cf. equation (7.12)). Alternatively, one can solve the following linear equation, obtained from rearranging equation (7.12) and including a normalization condition for $\mathbf{p}^*$ ($\sum_i p_i^* = 1$):

$$(T - I)\mathbf{p}^* + E\mathbf{p}^* = (1, 0, \dots, 0)^T. \quad (7.16)$$

Here E denotes a matrix that contains ones in the first row and is otherwise zero. After solving the set of linear equations, we obtain the stationary distribution of network states $\mathbf{p}^*$.

While the method to obtain the stationary distribution presented here is feasible for the analysis of small networks ($n < 20$), for example to study violation of detailed balance, inferences via constraints or the convergence times of Markov chains, the size of the transition matrix, and hence the computational effort, scales exponentially in the number of neurons. For larger systems, alternative approaches that consider only the lowest moments of the distribution p^* need to be considered (e.g., Helias, Tetzlaff, and Diesmann, 2014b; Dahmen, Bos, and Helias, 2016). In the next sections, we use this method to study sampling dynamics and inference via constraints in small networks for which no analytical form of the stationary distribution is known.

²M. Helias, pers. comm.

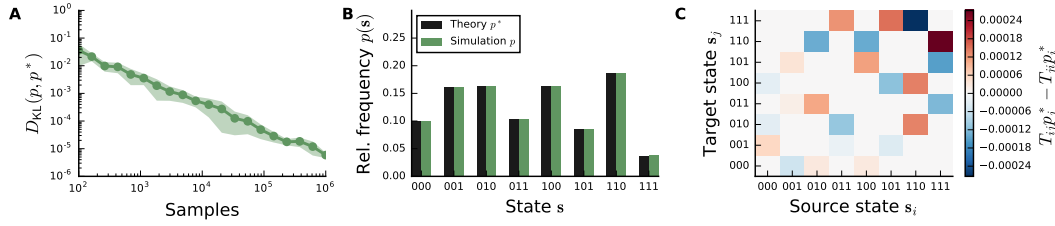


FIGURE 7.2: **Sampling networks with Gaussian noise.** **A** Sampling error $D_{\text{KL}}(p, p^*)$ between empirical distribution of network states $p(s)$ and theoretical stationary distribution $p^*(s)$ obtained via equation (7.16) as a function of the number of collected samples. Error bands represent mean $\pm$ SEM over 5 random network realizations. **B** Theoretical stationary distribution $p^*(s)$ (black) and empirical joint distribution $p(s)$ (green). **C** Net probability flux $T_{ij}p_j^* - T_{ji}p_i^*$ between state i and state j in stationary state.

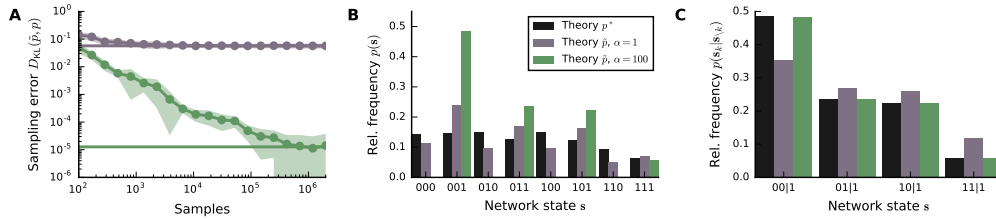


FIGURE 7.3: **Fast inference via constraints in sampling networks with Gaussian noise.** **A:** Sampling error $D_{\text{KL}}(\tilde{p}, p)$ between $\tilde{p}(s_k)$ with biases modified according to equation (7.7) and $p(s_k | s_{\setminus k})$ computed via equation (7.16) as a function of the number of collected samples. Error bands represent mean $\pm$ SEM over 5 random network realizations. Solid line indicates D_{KL} for infinite sampling duration. **B:** Marginal distribution $p(s)$ (black) and marginal of constrained distributions $\tilde{p}(s)$ with $\alpha = 1$ (dark purple) and $\alpha = 100$ (green), respectively, over the first three units. **C:** Conditional distribution $p(s_k | s_{\setminus k})$ and marginal of constrained distributions $\tilde{p}(s_k)$ for $\alpha \in \{1, 100\}$ over the unconstrained units.

7.3 Sampling networks with Gaussian noise

We now consider symmetric networks of deterministic binary units with additive Gaussian noise. To construct the transition matrix T , we compute the noise-averaged activation function (cf. section 5.2.4):

$$F_i(h_i) = \frac{1}{2} \operatorname{erfc} \left(-\frac{h_i + \mu_i}{\sqrt{2}\sigma_i} \right). \quad (7.17)$$

The stationary distribution $\mathbf{p}^*$ obtained from equation (7.16) using this activation function is approximated by a sampling network with units receiving Gaussian noise (figure 7.2). However, the net probability flux in the stationary state is nonzero (figure 7.2), indicating that the system does not obey detailed balance although the connections are symmetric. Similar observations were also reported by Crair and Bialek (1990) and Merolla, Ursell, and Arthur (2010) for networks of spiking neurons. This illustrates that not only asymmetric connectivity, but even deviations from the logistic form of the activation function can turn the network into a non-equilibrium system. In contrast to Boltzmann machines, one cannot efficiently compute conditional distributions in networks of Gaussian units by changing biases according to given evidence (equation (7.7)): The constrained marginal distribution does not match the exact conditional (figure 7.3). The deviations between the constrained

distribution and the exact conditional are, however, fairly small for the model considered here. It needs to be investigated in how far these deviations depend on the weights and the size of the sampling network, and whether they influence any real-world applications. These results nevertheless demonstrate that the same method routinely employed in Boltzmann machines, cannot be trivially transferred to other systems and that the particular form of stochasticity plays an important role.

7.4 The role of synaptic variability in neural computation

We now consider an alternative sources of stochasticity in neural systems: synaptic variability. Biological measurements demonstrate various form of variability intrinsic to the signaling via chemical synapses (Branco and Staras, 2009; Ribault, Sekimoto, and Triller, 2011a). For example, on the presynaptic side a single presynaptic spike can lead to varying numbers of vesicles fusing with the cell membrane to release neurotransmitter in the synaptic cleft. These vesicles can differ in the amount of neurotransmitter they contain. In addition, synaptic transmission can completely fail, with no postsynaptic response elicited by a presynaptic spike, observed quite frequently at synapses throughout the brain, with synaptic reliabilities of cortical neurons between 10 – 90% (Allen and Stevens, 1994; Branco and Staras, 2009). The combined effect leads to a significant variability in the observed post-synaptic potentials (Kandel, Schwartz, and Jessel, 1991; Faisal, Selen, and Wolpert, 2008a; Branco and Staras, 2009). This variability introduces a strong source of stochasticity in neural systems, and has recently received attention for its role in cortical computations (Rosenbaum, Rubin, and Doiron, 2012; Zhang and Peskin, 2015). Here we focus on exploiting synaptic stochasticity for approximate inference via sampling, as already suggested by Maass (2014), Hornschild (2014), Biermann (2016), Neftci et al. (2016), and Muller and Indiveri (2017).

We consider symmetric networks of deterministic binary units with Heaviside activation function. In the simple model considered here, synaptic stochasticity is introduced on the level of the input field via noise on the synaptic weights, mimicking variable PSPs observed in experimental studies:

$$h_i = \sum_j (w_{ij} + \xi_{ij}) s_j + b_i. \quad (7.18)$$

Here $\xi_{ij} \sim \mathcal{N}(0, \sigma^2)$ is uncorrelated Gaussian noise, assumed to have the same strength σ across all synapses in the network. Consistent with the absence of self connections ($w_{ii} = 0$), a neuron does not contribute to noise in its own input field and we assume $\xi_{ii} \equiv 0$. Note that the noise acts multiplicative, rather than additive as in the previous section. Extended models of synaptic variability alter the distribution of the multiplicative noise ξ_{ij} , for example, by considering synaptic failure (Neftci et al., 2016), scaling the noise strength with the strength of the synapse (Biermann, 2016) or by modeling multiple synaptic contacts with equal release probabilities (Kandel, Schwartz, and Jessel, 2000) via binomial distributions. We explicitly do not model temporal dynamics in the synapses, as e.g., short-term depression due to the finite recovery time of vesicles (e.g., Rosenbaum, Rubin, and Doiron, 2013) that contribute to observed variability. Instead, our model focuses on variability in post-synaptic responses that is not history dependent.

We compute the noise averaged activation function for this particular model similar to section 5.2.4. Note that in our model, even if a connection between to units

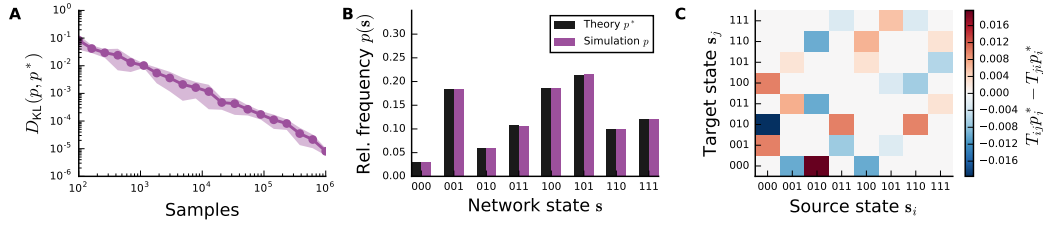


FIGURE 7.4: **Sampling networks with stochastic synapses.** **A** Sampling error $D_{\text{KL}}(p, p^*)$ between empirical distribution of network states p and theoretical stationary distribution p^* obtained via equation (7.16) as a function of the number of collected samples. Error bands represent mean $\pm$ SEM over 5 random network realizations. **B** Theoretical stationary distribution p^* (black) and empirical joint distribution p (purple). **C** Net probability flux $T_{ij}p_j^* - T_{ji}p_i^*$ between state i and state j in the stationary state.

is absent ($w_{ij} = 0$), the noise contribution ξ_{ij} is nonzero. This biophysically implausible feature can easily be overcome by scaling the noise amplitude with w_{ij} as in (Biermann, 2016), without fundamentally affecting our results. We integrate out the noise in every incoming synapse, leading to a multi-dimensional integral:

$$\begin{aligned}
 F_i(h_i) &= \int_{-\infty}^{\infty} d\xi_{i0} \cdots \int_{-\infty}^{\infty} d\xi_{in} \Theta \left(\sum_j (w_{ij} + \xi_{ij}) s_j + b_i \right) p(\xi_{i0} \cdots \xi_{in}) \\
 &= \int_{-\infty}^{\infty} d\xi_{i0} \cdots \int_{-\infty}^{\infty} d\xi_{in} \Theta \left(h_i^0 + \sum_j \xi_{ij} s_j \right) p(\xi_{i0}) \cdots p(\xi_{in}) \\
 &= \int_{-\infty}^{\infty} d\xi_i \Theta (h_i^0 + \xi_i) p(\xi_i) \\
 &= \frac{1}{2} \text{erfc} \left(-\frac{h_i^0}{\sqrt{2(|s| - \delta_{i+})\sigma}} \right). \tag{7.19}
 \end{aligned}$$

Here we used the pairwise-independence of ξ_{ij} and the properties of Gaussian RVs to replace $\sum_j \xi_{ij} s_j$ with $\xi_i \sim \mathcal{N}(0, (|s| - \delta_{i+})\sigma^2)$. The δ_{i+} arises due to the assumption that a neuron does not contribute noise to its own input field. We have introduced the input field in the absence of noise $h_i^0 := \sum_j w_{ij} s_j + b_i$. The main difference between equation (7.19) and the activation for Gaussian additive noise (equation (7.17)) is the presence of a term proportional to the number of active neurons in the network that effectively scales the noise strength σ . If we relate the noise strength to the inverse temperature β in Boltzmann machines, we observe that in networks with stochastic synapses the effective inverse temperature depends on the state of the network or, to be more precise, in our model on the total number of active neurons. The dependence of the effective noise strength on the network state forces us to consider networks in which at least one neuron has a positive bias, since otherwise the inactive network state becomes a stable fixed point and network activity dies out after a while.

As in the previous section, we use the noise-averaged activation function (equation (7.19)) to construct the transition matrix of the Markov chain and thereby obtain the stationary distribution p^* of network states. The empirical distribution of network states in networks with noise on the synaptic weights (cf. equation (7.18)) converges to the distribution obtained via the transition-matrix method (figure 7.4). Although the network is symmetric, we observe that the system does not exhibit

detailed balance and hence has a nonzero net probability flux between states even under stationary conditions. This is an effect of the state-dependent inverse temperature that leads to asymmetric effective interactions (Hornschild, 2014; Neftci et al., 2016). As for sampling networks with Gaussian noise, fast inference, i.e., sampling from the correct conditionals via constraining networks dynamics, is not possible for networks with stochastic synapses (Biermann, 2016). This severely limits their computational power.

Neftci et al. (2016) have demonstrated learning and inference with a network of stochastic synapses similar to the model presented above. In contrast to the binary units with states $s_i \in \{0, 1\}$ employed so far, they additionally consider bipolar units with states $s'_i \in \{-1, 1\}$. The state of a binary unit s_i can be translated into the state of a bipolar unit and vice versa:

$$s_i \leftrightarrow \frac{1}{2}(s'_i + 1). \quad (7.20)$$

While binary units in the off state have no influence on their targets, bipolar units in either state contribute to the input field of their targets. Intuitively, one should be able to compensate for the additional contributions in the off state by adjusting the biases of all units. To find a mapping between the parameters w, b of a network of binary units and parameters w', b' of a network of bipolar units, we require that Boltzmann machines with either type of state pairs sample from the same stationary distribution:

$$p(\mathbf{s}; w, b) \equiv p(\mathbf{s}'; w', b'). \quad (7.21)$$

For the energy for binary units we replace each s_i with the inverse transform from bipolar units (equation (7.20)) and reorder terms:

$$\begin{aligned} E(\mathbf{s}) &= -\frac{1}{2} \sum_{i,j} w_{ij} s_i s_j - \sum_i b_i s_i \\ &= -\frac{1}{2} \sum_{i,j} w_{ij} \frac{1}{2}(s'_i + 1) \frac{1}{2}(s'_j + 1) - \sum_i b_i \frac{1}{2}(s'_i + 1) \\ &= -\frac{1}{2} \sum_{i,j} \underbrace{\frac{1}{4} w_{ij}}_{=:w'_{ij}} s'_i s'_j - \sum_i \underbrace{\left(\frac{1}{8}(w_{ij} + w_{ji}) + \frac{b_i}{2} \right)}_{=:b'_i} s'_i + C. \end{aligned}$$

The additive constant C is irrelevant for the evaluation of state probabilities, since it just represents a global offset in energies that cancels with an identical term appearing in the normalization constant. We hence obtain the following mapping for the parameters w and b :

$$w_{ij} \rightarrow \frac{1}{4} w_{ij} \quad (7.22)$$

$$b_i \rightarrow \frac{1}{8}(w_{ij} + w_{ji}) + \frac{b_i}{2} \quad (7.23)$$

Using these parameters for a network of bipolar units, the joint distribution of network states has Boltzmann form ($p(\mathbf{s}) \sim e^{-\beta E(\mathbf{s})}$) with identical probabilities as in

the binary case. By further requiring that the systems obeys detailed balance (equation (7.13)), we derive the activation functions for single units necessary for implementing the corresponding sampling dynamics:

$$\begin{aligned} \frac{p(s_i = 1|\mathbf{s}^{i-})}{p(s_i = -1|\mathbf{s}^{i+})} &= \frac{p(\mathbf{s}^{i+})}{p(\mathbf{s}^{i-})} \\ &= \frac{\exp\left(\beta \sum_j w_{ij}s_j + b_i + C\right)}{\exp\left(-\beta \sum_j w_{ij}s_j - b_i + C\right)} \\ &= e^{2\beta h_i}, \end{aligned}$$

with $h_i = \sum_j w_{ij}s_j + b_i$. Since the transition probability of a unit is independent of the unit itself, we can use the normalization condition to obtain

$$\frac{p(s_i = 1|\mathbf{s}^{i-})}{p(s_i = -1|\mathbf{s}^{i+})} = \frac{p(s_i = 1|\mathbf{s})}{1 - p(s_i = 1|\mathbf{s})},$$

leading to

$$\begin{aligned} \frac{p(s_i = 1|\mathbf{s})}{(1 - p(s_i = 1|\mathbf{s}))} &= e^{2\beta h_i} \\ \Rightarrow p(s_i = 1|\mathbf{s}) &= \frac{1}{1 + e^{-2\beta h_i}}. \end{aligned}$$

In contrast to binary units an additional factor 2 appears in the exponent of the exponential.

With this translation of parameters and the activation function for bipolar units, we can systematically investigate the differences in network dynamics for the different types of on and off states. Neftci et al. (2016) argue:

“[...], we show that S2Ms with activation values (-1,+1) are similar to Boltzmann machines, except that the activation function of the neuron is an error function instead of the logistic function, and where the input is invariant to rescaling.”,

where “S2M” refers to a similar type of network model that we consider here, with an identical noise-averaged activation function. We investigate this claim, and in particular their subsequent assumption about the extension of their initial results from bipolar to binary units:

“We assumed that the S2M with activation levels (0,1) and the spiking S2M are sufficiently similar to the Boltzmann distribution [...]”.

In the following we hence consider four different sampling networks:

- (i) a Boltzmann machine with binary (0, 1) states
- (ii) a network of deterministic units with Gaussian additive noise with binary (0, 1) states
- (iii) a network of deterministic units with Gaussian noise on the synaptic weights with binary (0, 1) states
- (iv) a network of deterministic units with Gaussian noise on the synaptic weights with bipolar (−1, 1) states

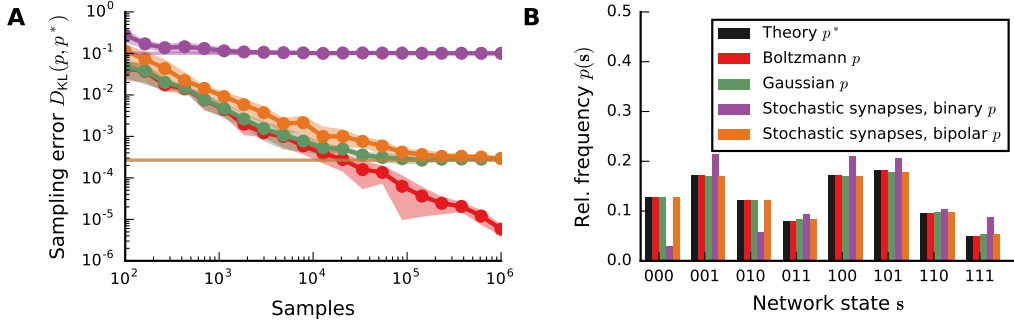


FIGURE 7.5: **Approximating Boltzmann distributions with different types of noise.** **A:** Sampling error $D_{KL}(p, p^*)$ between empirical distribution of network states p and theoretical stationary distribution p^* obtained via equation (5.3) as a function of the number of collected samples. Error bands represent mean $\pm$ SEM over 5 random network realizations. **B:** Theoretical stationary distribution p^* (black) and empirical joint distribution p for Boltzmann machines (red), networks with Gaussian additive noise (green), networks with stochastic synapses and binary units (purple), and networks with stochastic synapses and bipolar units (orange).

We investigate how the latter three networks differ in their ability to approximate Boltzmann machines and compare our results to the claims above. We already introduced parameter mappings for the case (ii) and (iii) (cf. equation (7.19) and equation (5.9), respectively) and now we seek a similar expression for case (iv). As for sampling networks with binary units and stochastic synapses, we consider additive noise on the synaptic weights with a Gaussian distribution (equation (7.18)). Similar to Neftci et al. (2016) and the derivation of equation (7.19) the noise averaged activation function for bipolar units with stochastic synapses is under these conditions given by

$$F(h_i) = \frac{1}{2} \operatorname{erfc} \left(-\frac{h_i}{\sqrt{2(n-1)\sigma}} \right). \quad (7.24)$$

Here n denotes the number of units in the network and the -1 comes from the absence of self connections, which implies $\xi_{ii} = 0$. Note that equation (7.24) is independent of the current network state in contrast to the case of binary units equation (7.19), implying that the inverse temperature is constant. We further observe that we can rescale $\sigma \rightarrow \sqrt{\frac{1}{n-1}}\sigma$ to make the activation functions for binary units with Gaussian additive noise and bipolar units with synaptic noise identical.

Under this rescaling of parameters, the stationary distribution of networks with Gaussian noise and networks with stochastic synapses and bipolar states becomes identical and both can approximate a Boltzmann distribution fairly well, however, with a finite error even for long sampling duration (figure 7.5). In contrast, for networks of binary units with stochastic synapses, the deviation between the target and the sampled distribution is substantial (figure 7.5). Most prominently, the inactive network state $((0, \dots, 0)^T)$ is severely misrepresented due to state-dependent noise in networks of binary units. This demonstrates that in contrast to Neftci et al.'s claims, networks with binary and bipolar units differ substantially in their sampling dynamics and hence their stationary distributions of network states. Further careful and systematic investigation is necessary to clarify these issues and to understand why a straightforward application of CD learning as in Neftci et al. (2016) nevertheless can be successfully used to train a network with stochastic synapses to classify

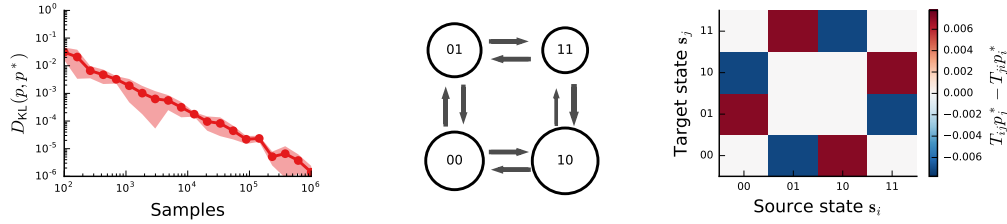


FIGURE 7.6: **Asymmetric sampling networks.** **A** Sampling error $D_{KL}(p, p^*)$ between empirical distribution of network states p and theoretical stationary distribution p^* obtained via equation (7.16) as a function of the number of collected samples. Error bands represent mean $\pm$ SEM over 5 random network realizations. **B** Fluxes in a non-equilibrium network of two neurons. Size of circles represent stationary probability of the respective states. Arrows have width proportional to transition probabilities $T(s_j, s_i)$. **C** Net probability flux $T_{ij}p_j^* - T_{ji}p_i^*$ between state i and state j in stationary state.

handwritten digits. One possibility is that the behavior of the two models becomes sufficiently similar in the limit of large networks. In these, the activity is typically constrained to a small region of state space, approximately determined by the mean and variance of the population activity, over which variations in temperature are possibly negligible. We conclude that the sampling dynamics in cases (ii)-(iv) differ substantially from Boltzmann machines and based on our previous results argue that fast inference via constraining of sampling dynamics is not exact in any of them.

7.5 Sampling in asymmetric networks

In contrast to the symmetric sampling networks considered so far, biological neuronal networks are asymmetric. The asymmetry arises fundamentally due to the separation into excitatory and inhibitory neurons, but also due to the locality of synaptic plasticity: even bi-directionally connected neurons will maintain different strengths of their reciprocal connections. In the presence of asymmetric connections, networks generally have a non-vanishing probability flux in state space even under stationary conditions. While this is expected to provide additional computational power, under these circumstances it is generally not possible to derive a closed-form expression for the distribution of network states.

We consider sampling networks with random weight matrices, in particular with $w_{ij} \neq w_{ji}$, random biases b_i and logistic activation functions. By constructing the transition matrix T we can compute the stationary distribution for such a network (figure 7.6). In these non-symmetric networks we observe a strong violation of the detailed-balance condition (figure 7.6) – for systems obeying detailed balance, the net probability flux $T_{ij}p_j^* - T_{ji}p_i^*$ would be identical to zero.

Again, we consider an inference problem: given the states of a subset of units, compute the conditional distribution over the remaining units ($p(s_k | s_{\setminus k})$). As in Boltzmann machines, we implement the evidence by constraining network dynamics through manipulation of biases (equation (7.7)) and observe the marginal over the unconstrained units ($\tilde{p}(s_k)$). The exact conditionals and the constrained marginals exhibit significant deviations (figure 7.7), demonstrating that the same approach that works very well in Boltzmann machines does not lead to the desired results in asymmetric networks. When extending the sampling framework to asymmetric networks, one needs to further investigate in how far the useful feature of Boltzmann machines to allow fast inference via constraining network dynamics

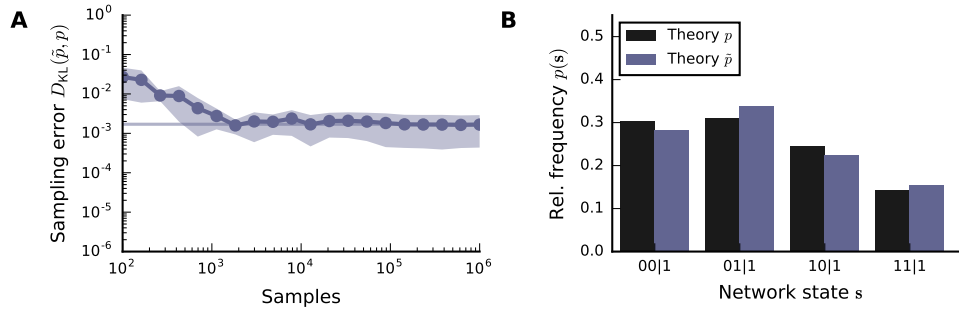


FIGURE 7.7: **Fast inference via constraints in asymmetric sampling networks.** **A:** Sampling error $D_{KL}(\hat{p}, p)$ between empirical marginal distribution of network states $\hat{p}$ with biases modified according to equation (7.7) and theoretical conditional distribution p computed from equation (7.16) as a function of the number of collected samples. Error bands represent mean $\pm$ SEM over 5 random network realizations. Solid line indicates D_{KL} for infinite sampling duration. **B:** Conditional distribution $p(s_k|s_{\setminus k})$ and marginal of constrained distribution $\hat{p}(s_k)$.

can be retained. Our results demonstrate that a naive implementation of evidence comes at a significantly loss of accuracy in representing the conditional distribution.

7.6 Discussion

The work presented here is a further step towards biophysically plausible implementations of approximate inference in neural substrates. We initially demonstrated how fast inference is possible in Boltzmann machines by constraining network dynamics, forcing the network to generate samples directly from the exact conditionals. While for Boltzmann machines and their modern spiking counterparts (Buesing et al., 2011), the exact distribution of networks states is known, this is currently not the case for all alternative approaches. In particular, it is often assumed that inference in alternative models is as easily possible as in Boltzmann machines (Neftci et al., 2016; Muller and Indiveri, 2017), most likely arising from the view that the stochasticity in Boltzmann machines is only required to introduce some form of variability that prevents symmetric networks to lock into fixed states like Hopfield networks (Hopfield, 1982b). We have demonstrated here that in contrast to this assumption the exact distribution of the noise is highly relevant – deviations from the particular logistic activation function already lead to the loss of sampling networks to efficiently sample from conditionals by constraining their network dynamics. Our findings provide strong evidence that similar computational shortcuts as in Boltzmann machines, are not trivially translatable to any other of the sampling approaches considered here. It should be noted that we do not question the ability of any of these models to represent or possibly learn well-defined probability distributions, but rather the process of performing inferences via constraints in the network dynamics which gives Boltzmann machines their computational power.

Networks with stochastic synapses have recently generated interest for neuromorphic implementations (Neftci et al., 2016; Muller and Indiveri, 2017), also since they can possibly be implemented with recently developed phase-change materials. These materials hold strong potential for the implementation of energy- and space-efficient plastic synapses, but are inherently stochastic (see, e.g., Bill and Legenstein, 2014; Serb et al., 2016). The desire to interpret this stochasticity as a computational resource in sampling networks is understandable, however, we caution against the

view that sampling networks with stochastic synapses are “sufficiently similar” to Boltzmann machines until the source of deviations during inferences is sufficiently understood and argue that for future neuromorphic applications, more detailed investigations of these models is necessary.

We now leave the theoretical domain and turn to the technical aspects of the numerical simulation of neural network models. The small models investigated so far in this thesis can be conveniently simulated on modern laptops and workstations. To investigate large-scale network models with millions to billions of individual neurons, however, researchers need to take advantage of high-performance computing facilities. In the next chapter we discuss simulation technology that makes efficient use of these systems.

Chapter 8

Extremely scalable spiking neural network simulation code: from laptops to exascale computers

This chapter is based on the following publication:

Jakob Jordan, Tammo Ippen, Moritz Helias, Itaru Kitayama, Mitsuhsa Sato, Jun Igarashi, Markus Diesmann, and Susanne Kunkel (2017a). “Extremely scalable spiking neural network simulation code: from laptops to exascale computers”. (submitted)

Author contributions:

Under supervision of Moritz Helias, Markus Diesmann and Susanne Kunkel, the author performed work on all parts of the concepts of the new simulation technology, its implementation in the NEST kernel, and the data analysis and interpretation. All authors contributed to the writing of the manuscript.

8.1 Introduction

Modern neuroscience has established numerical simulation as a third pillar supporting the investigation of the dynamics and function of neural networks, next to experimental and theoretical approaches. Simulation software reflects the diversity of modern neuroscientific research with tools ranging from the molecular scale to investigate processes at individual synapses (Wils and De Schutter, 2009) to whole-brain simulations at the population-level that can be directly related to clinical measures (Sanz Leon et al., 2013). Most neuronal-network simulation software, however, is based on the hypothesis that the main processes of brain function can be captured at the level of individual nerve cells and their interactions through electrical pulses. Since these pulses show little variation in shape, it is generally believed that they convey information only through their timing or rate of occurrence. Simulators in this area that follow a general-purpose approach employ simplified models of neurons and synapses with individually configurable parameters and connectivity (Carnevale and Hines, 2006; Bower and Beeman, 2007; Gewaltig and Diesmann, 2007; Goodman and Brette, 2013; Bekolay et al., 2013). In such simplified models, individual neuron and synapse dynamics are typically described by a small number of coupled differential equations. Besides, in order to address models of learning in neural networks most simulators support a variety of plasticity mechanisms such as short- and long-term plasticity (Morrison, Diesmann, and Gerstner, 2008), neuromodulated plasticity (Potjans, Morrison, and Diesmann, 2010) and structural plasticity (Diaz-Pier et al., 2016).

It is widely believed that high-level brain function is not solely the product of complex dynamics of isolated brain areas, but involves coordinated interaction between multiple cortical and subcortical areas (Kandel, Schwartz, and Jessel, 1991; Bressler and Menon, 2010). To gain insights into cortical information processing network models hence need to account for several areas and their interactions involving millions of neurons and billions of synapses. A systematic reduction in neuron and synapse density compared to biological tissue is severely limited as soon as researchers strive to faithfully represent even just pairwise coordinated neuronal activity (Albada, Helias, and Diesmann, 2015), making the need for full-scale models apparent. While workstations allow simulations of up to 10^5 neurons, corresponding to the number of neurons under approximately 1 mm^2 of cortical surface, larger networks require distributed simulations (Senk et al., 2015; Schmidt et al., 2016). State-of-the-art simulation software allows researchers to simulate about 10% of the human cortex at a resolution of individual neurons and synapses on contemporary supercomputers (Kunkel et al., 2014). For these large-scale simulations one of the main computational challenges is the high connectivity of neuronal networks. The human cortex consists of about 10^{10} cells, each receiving about 10^4 connections (Abeles, 1991; Stepanyants et al., 2009), which leads to an estimated 10^{14} synapses. Representing each of the connections by two double precision numbers requires about 1.6PB of main memory. To complicate things further, neurons receive only about 50% of their connections from other nerve cells in their vicinity, while the remainder are long-range connections from various remote areas (Abeles, 1991; Braitenberg and Schüz, 1991). This feature distinguishes neuronal simulations from simulations of classical physical systems, for example by finite-element methods (see, e.g., Johnson, 1987) exploiting the locality of physical interactions, and poses severe difficulties for dynamic load balancing: One cannot easily move computational units from one process to the next without major changes to the fundamental data structures.

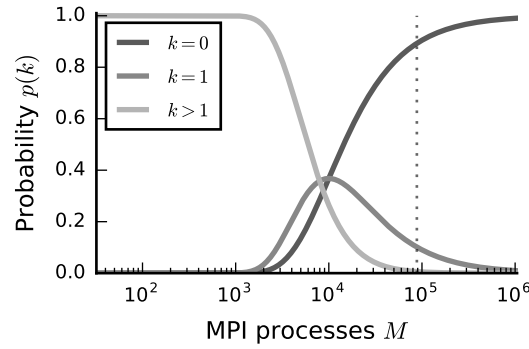


FIGURE 8.1: **Probability of a model neuron to have a certain number of targets on a randomly selected process in a weak scaling scenario.** Probability of a neuron to have zero (dark gray), one (medium gray), or more (light gray) targets. Vertical dashed line marks maximal number of MPI processes on the K computer. Targets per neuron $K = 10^4$, number of MPI processes $M \in [32; 1,048,575]$.

Over a decade ago simulation codes started to store the data that represent synapses exclusively on the compute node where the target neuron resides (Morrison et al., 2005), in the following referred to as the postsynaptic side. For typical neuronal network models this approach enables parallel network construction with none or little communication between the compute nodes. Furthermore, in this scheme only the occurrence of a spike in a particular source neuron needs to be transmitted to other compute nodes, often in source-based address-event-representation (AER; Boahen, 2000; Lansner and Diesmann, 2012b). Typically `MPI_Allgather` is employed to gather all spikes generated in the network on each compute node (Hines, Kumar, and Schürmann, 2011). Postsynaptic data structures are then responsible for obtaining the relevant synapse parameters from local memory and routing the spike to the correct target. To distribute workload evenly across compute nodes also for highly structured networks with heterogeneous population properties, neurons are distributed across MPI processes in a round-robin fashion. On small machines the number of compute nodes participating in a simulation is smaller than the number of synapses per neuron such that each neuron typically has many targets on every compute node. In this setting it is efficient to maintain on each compute node a vector containing references to local target lists with one element for each neuron in the network that could be indexed by the source neurons identifier (Morrison et al., 2005). With growing network size and the availability of a new generation of supercomputers, the ratio between the number of compute nodes and the number of synapses per neuron reversed. This led to a replacement of the vector by a sparse table with subsequent target lists implemented as dynamic container types that have little overhead for a small number of local targets. The sparse table accounts for the sparsity of large-scale networks by consuming only few bits of memory for each empty entry (Kunkel et al., 2014). These data structures enabled simulation codes that perform equally well on small- to large-scale simulations, and scale well to the largest supercomputers available today (“petascale” regime, 10^{15} FLOPS) (Kunkel et al., 2014). Over the next decade, exascale machines will be developed that will most likely show only a moderate increase in the number of compute nodes but a significant increase the number of threads and amount of memory per compute node (see, e.g., Dongarra et al., 2011). If neural simulators are able to fully exploit the computational power of these new machines, researchers, for the first time, will be able to simulate the full human cortex at cellular resolution.

In the case of purely postsynaptic storage of connection information, all spikes from the network are collected on all compute nodes since only postsynaptic data structures are responsible for routing the spikes to the correct targets. To support efficient delivery of spikes, this requires the postsynaptic data structures to store at least one bit per neuron in the network signaling the presence or absence of local targets (Kunkel et al., 2014). In this case memory usage per compute node scales with the total number of neurons in the network, consuming a major part of the available local memory from about 10^9 to 10^{10} neurons on. However, networks in which each neuron receives a fixed number of connections become extremely sparse at this scale. Assuming a fixed number of neurons per process and 10^4 targets per neuron, most neurons have multiple targets on every process for less than 10^3 MPI processes (figure 8.1, light gray). This probability quickly decays to zero between 10^3 and 10^4 processes as it becomes more likely to find exactly one target per process (figure 8.1, medium gray). Beyond 10^4 MPI processes it is overwhelmingly likely that a specific neuron has no target on a randomly selected process (figure 8.1, dark gray), implying that most spikes collected on a compute node via `MPI_Allgather` do not have any local targets (see also Hines, Kumar, and Schürmann, 2011). This estimate demonstrates that a communication scheme that relies purely on postsynaptic routing of spikes cannot be efficient in the regime of tens of thousands of MPI processes. Nevertheless, the probability that at least one neuron on a randomly selected MPI process has local targets on another randomly selected process, is not small. Consequently all processes potentially need to communicate spikes to each other at some point during a simulation, ruling out MPI communication schemes that only include subsets of nodes.

In this chapter we describe a two-tier connection infrastructure with a pre- and postsynaptic part that replaces the above mentioned purely postsynaptic storage of connection information and enables directed communication among compute nodes via `MPI_Alltoall`. In particular we replace all data structures that scale proportional to the total number of neurons in the network or the total number of MPI processes, because they consume significant amounts of memory for simulations of networks of 10^9 neurons or more. While there is need for such large-scale simulations, many researchers investigate small to medium size networks. Therefore, the new data structures and communication framework should not lead to a penalty for simulations run on laptops, workstations, and moderately sized clusters. We consequently introduce additional optimizations to maintain high performance in small-to medium-scale simulations. This approach preserves a common codebase for laptops and HPC systems, reducing maintenance costs and supporting quick adoption of features designed for either use case.

The remainder of this chapter is organized as follows: In section 8.2.1 we describe an archetypal network model considered as the main use case in this chapter. In section 8.2.2 we introduce the three scaling scenarios that we consider here to assess performance. The chapter continues in section 8.2.3 by describing the three supercomputing platforms employed in this study. Section 8.2.4 gives a short overview of the NEST simulator for which we provide an implementation of the new connection infrastructure and communication scheme. To predict the influence of the new data structures on memory consumption we describe a model that estimates the memory usage per compute node in section 8.2.5. At the beginning of section 8.3 we discuss the limitations of the connection infrastructure implemented in the previous kernel, henceforth referred to as ‘4g’, and afterwards describe the new two-tier infrastructure and corresponding network construction procedure employed in the new simulation kernel (‘5g’) in section 8.3.1. We continue by describing the new

spike exchange method in section 8.3.2. Afterwards we consider small-scale simulations and introduce corresponding optimizations in section 8.3.3. We finally discuss the results obtained from the memory-usage model in section 8.3.4 and from performance measurements in various scaling scenarios (section 8.3.5). We conclude the chapter in section 8.5 by discussing limitations of the new technology and future extensions. The technology described in this chapter will be made available to the community with one of the next major releases of the open-source simulation software NEST. The conceptual and algorithmic work described here is a module in our long-term collaborative project to provide the technology for neural systems simulations (Gewaltig and Diesmann, 2007).

8.2 Materials and Methods

8.2.1 Benchmark network model

To analyze the memory usage of the new data structures (see section 8.2.5) and to measure the actual memory usage and run time of the implementation (see section 8.2.2) we employ a balanced random network model with plastic connections, also used in previous publications on neuronal network simulation technology (Helias et al., 2012; Kunkel et al., 2012a; Kunkel et al., 2014; Ippen et al., 2017; Kunkel and Schenck, 2017a). The network consists of two recurrently connected populations: one excitatory and one inhibitory, where excitatory neurons outnumber inhibitory neurons by a factor of four. To ensure stability of the network the inhibitory connections are much stronger than the excitatory connections. Neurons are modeled by single-compartment leaky-integrate-and-fire neurons with alpha-shaped postsynaptic currents and have homogeneous parameters within and across the two populations. Connections are drawn randomly for each neuron with a fixed number of incoming connections per neuron independent of the network size. The excitatory-excitatory connections exhibit spike-timing dependent plasticity (STDP, see, e.g., Morrison, Aertsen, and Diesmann, 2007a), while all other connections are static. This model serves as a scalable version of a typical neuronal network simulation. The results obtained with this particular model generalize, as long as the total memory usage of synapses is significantly larger than that of neurons, and interactions are mainly mediated via chemical synapses. Due to its random connectivity the network model represents a worst-case scenario in terms of network structure: A single neuron projects with equal probability to any other neuron in the network such that local communication patterns cannot be exploited by representing strongly connected subnetworks on a subset of the available compute nodes. A detailed network description and parameter values can be found in section C.1 and a variant of the simulation script is available from the most recent NEST release as `hpc_benchmark.sli` (Kunkel et al., 2017).

8.2.2 Measuring scalability

To assess the scalability of the simulation code across MPI processes and threads, we investigate three scaling scenarios. For an in-depth discussion of the relevant scaling scenarios and possible pitfalls, see Albada et al. (2014). Since we run a single MPI process per compute node in all simulations of this study, we use “MPI processes” and “compute nodes” interchangeably in this chapter.

In weak-scaling benchmarks, the problem size per compute node is fixed while the number of compute nodes and thus the total problem size is varied. In our

case, we simulate a constant number of neurons per compute node where all neurons have a fixed in-degree, which leads to an increase in network size with sparser connectivity as the number of compute nodes grows. A weak-scaling experiment uncovers limiting factors for scalability in terms of memory usage and runtime that increase proportionally with the total network size ($\sim N$) or the number of MPI processes ($\sim M$).

In strong-scaling measurements, the total problem size is fixed while the number of MPI processes or threads per process is varied. Here, we fix the total network size including the number of connections. With an increasing number of MPI processes or threads per process, this reduces the load per process or thread, addressing the question of how fast a network of a particular size can be simulated. While in our application a strong-scaling test over MPI processes uncovers communication bottlenecks, a strong-scaling test over threads mainly exposes serial parts of the code.

In addition to weak-scaling and strong-scaling experiments, we perform a maximum-filling scaling. For a given amount of computational resources, in terms of available memory per compute node, we determine the maximal problem size that can be simulated. This is not necessarily identical to a weak scaling if, for example, the memory usage of the application changes with the number of MPI processes. In our case, we determine the maximal number of neurons that just fits into the available memory for a given number of compute nodes where all neurons have a fixed in-degree. Since the maximum network size is difficult to determine, we obtain a prediction from the memory-usage model (see section 8.2.5) before performing a full-scale run. The maximum-filling scaling scenario tests the limits of the software in terms of memory usage and addresses the issue of efficient use of available computational resources: How many compute nodes does a specific network simulation require at least?

8.2.3 Supercomputers

We run benchmarks on three HPC systems that are commonly employed for (neuro) scientific research: the JUQUEEN BlueGene/Q and JURECA systems at the Jlich Research Centre, Germany, and the K computer at the Advanced Institute for Computational Science in Kobe, Japan.

The JUQUEEN supercomputer (Jülich Supercomputing Centre, 2015) consists of 28,672 compute nodes, each equipped with a 16-core IBM PowerPC A2 processor running at 1.6GHz and 16GB RAM, leading to a peak performance of about 5.9PFLOPS and 448TB of main memory in total. The communication network is implemented as a 5D torus with a bandwidth of 40GBps. Applications are compiled using the IBM XL compiler suite. JUQUEEN supports hybrid parallelism, with multithreading via OpenMP¹ within a single compute node and MPI² for distributed-memory computing. The GNU Scientific library³ is available in version 2.1.

JURECA consists of 1872 compute nodes, each housing two Intel Xeon E5-2680 v3 Haswell CPUs at 2.5GHz for a total of 1.8PFLOPS. Most of the compute nodes have 128GiB of memory available. The system provides 75 compute nodes equipped with two NVIDIA K80 GPUs, which, however, were not used in this study. Nodes are connected via Mellanox EDR InfiniBand. To compile applications, we rely on the GNU Compiler Collection (GCC) and link against the ParaStationMPI library for MPI support.

¹v3.1, <http://www.openmp.org/specifications/>

²v2.1, <http://www.mpi-forum.org/docs/docs.html>

³<https://www.gnu.org/software/gsl/>

The K computer (Miyazaki et al., 2012) features 82,944 compute nodes, each with an 8-core SPARC64 VIIIfx processor operating at 2GHz, with 16GB RAM/node, leading to a peak performance of about 11.3PFLOPS and a total of 1377TB of main memory. The compute nodes are interconnected via the “Tofu” (“torus connected full connection”) network with 5GBps per link. The K computer supports hybrid parallelism with OpenMP (v3.0) at the single node level and MPI (v2.1) for inter-node communication. Applications are compiled with the Fujitsu C/C++ Compiler.

8.2.4 NEST simulator

NEST is an open-source software tool that is designed for the simulation of large-scale networks of single-compartment spiking neuron models (Gewaltig and Diesmann, 2007). It is developed and maintained by the NEST initiative⁴ under the GNU General Public License, version 2⁵ and can be freely downloaded from the website of the NEST simulator⁶. The collaborative development of NEST follows an iterative, incremental strategy derived from the requirements and constraints given by the community (Diesmann and Gewaltig, 2002). Users can control simulations either via a built-in scripting language (SLI) or a Python module (PyNEST; Eppler et al., 2009). While the definition of the network, in terms of the specification of neuronal populations and connections, can be conveniently performed in procedural form in an interpreted language, all compute-intensive tasks such as the actual generation of connectivity or the propagation of neuron dynamics are executed by the simulation kernel implemented in C++. NEST supports a wide variety of computing platforms, from laptops to moderately-sized clusters and supercomputers using a common codebase. To optimally use the available compute resources, NEST supports hybrid parallelization employing MPI for inter-node communication and multi-threading via OpenMP within each MPI process. Running multiple threads instead of multiple MPI processes per compute node makes better use of the available memory (Ippen et al., 2017).

Neurons are distributed in a round-robin fashion across all available threads according to their global id (GID), which labels all neurons and devices in the network uniquely by order of creation. The round-robin distribution of neurons implements a simple form of static load balancing as it assures that neurons which belong to the same population and are hence expected to exhibit similar activity patterns, are evenly distributed across cores. Devices for stimulation and recording, are duplicated on each thread and only send to or record from thread-local neurons to avoid expensive communication of status variables. Events between neurons are communicated between processes by collective MPI functions (see section 8.3.2). Most data structures are private to each thread within a compute node. This separation is however relaxed during writing of events to MPI buffers and reading of events from the buffers to improve efficiency and reduce serial overhead (see section 8.3.1 and section 8.3.2). NEST offers a range of neuron and synapse models from low to high complexity. Users can extend the range of available models by employing a domain-specific model description language (Plotnikov et al., 2016b) or by providing an appropriate implementation in C++. The simulation kernel supports further biophysical mechanisms, for example neuromodulated plasticity (Potjans, Morrison, and Diesmann, 2010), structural plasticity (Diaz-Pier et al., 2016), coupling between

⁴<http://nest-initiative.org/>

⁵<https://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

⁶<http://www.nest-simulator.org/>

neurons via gap junctions (Hahne et al., 2015), and non-spiking neurons with continuous interactions, such as rate-based models (Hahne et al., 2017).

8.2.5 Adaptation of memory-usage model

Improving algorithms and data structures within an existing software project requires first of all identification of the main bottlenecks. This involves measurements of runtime and memory usage (see, e.g., Hager and Wellein, 2011), since any intuitions about possible bottlenecks can be severely misleading (see, e.g., Bentley, 1982). This is ever more the case when redesigning algorithms and data structures that need to scale to tens of thousands of processes. Measurements for large-scale applications, however, consume time and resources. Kunkel et al. (2012a) therefore introduce a model that allows the prediction of the memory usage of a neural simulator accounting for contributions from different objects, such as neurons, synapses, and the corresponding infrastructure. The model considers only the leading order contributions to the overall memory consumption and needs to be checked against actual measurements to prove its sufficiency.

The model describes the memory consumption per MPI process as a function of compute environment and network parameters such as the number of MPI processes, the number of threads per process, and the total number of neurons in the network. Applying this model to NEST as a concrete use case allows us to predict the effect of potential changes to code quickly, without running simulations, and hence to target our efforts on critical parts of the codebase. In addition it enables us to determine the limits (in terms of network size) for a particular amount of compute resources (see section 8.2.2). For extensive discussions of the memory-usage model, please refer to Helias et al. (2012) and Kunkel et al. (2012a). See Kunkel et al. (2014) for the memory model describing the previous simulation kernel.

Here we present adaptations of the memory-usage model that allow us to apply the model to the new kernel. Originally, the model was defined as a function of the number of MPI processes M , the number of threads per MPI process T , the total number of neurons N and the average number of connections per neuron K . Since we keep K fixed throughout this study, we will describe the memory usage as a function of only three variables: $\mathcal{M}(M, T, N)$. Please refer to section C.1 for the numerical values of all other parameters that appear in the equations below. The total memory usage can be divided into three components: base memory usage and MPI buffers $\mathcal{M}_0(M, T, N)$, memory usage of nodes $\mathcal{M}_n(M, N)$, and memory usage of connections $\mathcal{M}_c(M, T, N)$. The latter two components do not just contain the memory usage of individual neuron and synapse objects, but also contributions from infrastructure needed for efficient access to the individual objects during simulation. This leads to the following definition of the full model (cf. Kunkel et al., 2014):

$$\mathcal{M}(M, T, N) = \mathcal{M}_0(M, T, N) + \mathcal{M}_n(M, N) + \mathcal{M}_c(M, T, N) .$$

The first term contains the empirically measured base memory usage $\mathcal{M}_b$, including the memory required by the simulation kernel just after startup, MPI and OpenMP overhead. Additionally this factor captures the memory usage of MPI buffers for the communication of connections (see section 8.3.1) and spike events during simulation. This leads to the following definition of $\mathcal{M}_0$:

$$\mathcal{M}_0(M, T, N) = \mathcal{M}_b + \min(B_c, N_M \min(K, MT))m_{td} \\ + \min(B_s, N_M \nu_{\max} \min(K, MT))m_{sd} ,$$

where we introduced the shorthand $N_M := \frac{N}{M}$. Here m_{td} denotes the memory consumption for a single entry in the MPI buffer used for communication of connectivity data (section 8.3.1) and m_{sd} denotes the memory consumption for a single entry in the MPI buffer used for communication of spikes (section 8.3.2). Since we employ `MPI_Alltoall`, the buffer size for a single communication round must be the same across all ranks. If more data need to be communicated than a single communication round can handle, we initiate another round of collective communication and double the size of the respective buffers, up to a user-defined maximal size, denoted by B_c and B_s , respectively (cf. section 8.3.1, section 8.3.2). The (average) number of connections and spikes can be estimated as $N_M \min(K, MT)$ and $N_M \nu_{\max} \min(K, MT)$, respectively, where $\nu_{\max}$ denotes the maximal firing rate in the network. We assume that a single neuron has an average out-degree of K , independent of the size of the network as described in section 8.2.1 and consistent with biological data (Abeles, 1991). The occurrence of the total number of threads (MT) in $\min(K, MT)$ is due to adaptations for pre-petascale machines as described in section 8.3.3.

As the redesign affects only the connection infrastructure and communication framework, the contributions of neurons and neuronal infrastructure are the same as in the memory-usage model for the previous kernel. For a definition and discussion of this second contribution $\mathcal{M}_n(M, N)$ to the overall memory consumption please refer to Kunkel et al. (2014).

In the previous simulation kernel, all information about connections was exclusively stored on the postsynaptic side, which is the compute node on which the target neuron resides. The main difference in the memory model for the new kernel is an additional term that describes the presynaptic part of the connection infrastructure, storing the location of all targets of the local neurons. The contributions of connections and corresponding infrastructure in the new kernel is described by

$$\mathcal{M}_c(M, T, N) = K_M^{\text{stat}} m_c^{\text{stat}} + K_M^{\text{stdp}} m_c^{\text{stdp}} + K N_M m_s + \frac{N}{M} \max(K, MT) m_t .$$

The first two components describe the memory consumption of the actual synapse objects (m_c^{stat} and m_c^{stdp} represent the memory usages of a single static or STDP synapse, respectively), the third term is the contribution of the data structure storing the sources of the respective connections (m_s is the memory consumption of a single source, see section 8.3.1), and the fourth term represents the connection information on the presynaptic side (with m_t the memory consumption of a single target, see section 8.3.1). $K_M^{\text{stat}} + K_M^{\text{stdp}} = K N_M$ is the total number of local synapses, including all static and plastic synapses.

8.3 Results

Before presenting the new simulation kernel, we shortly discuss the main bottlenecks of the present technology for large-scale neural networks simulations. It was previously suggested that synapse objects should be stored on the compute node on which their target neuron resides (Morrison et al., 2005). On the one hand, this

choice reduces the amount of information to be communicated during simulation of the network, while on the other hand it allows for completely parallel network construction, a practical necessity for large-scale simulations. The previous kernel of NEST uses a source-based address-event-representation (AER) scheme (Boahen, 2000; Lansner and Diesmann, 2012b), employing `MPI_Allgather` to communicate spike events among compute nodes: Each MPI process receives the global ids (GIDs) of all neurons that emitted a spike in the last communication interval, and each thread needs to determine the thread-local neurons to which it needs to deliver the events. To this end, each thread is equipped with a sparse table for efficient lookup of connection objects via the GID of the source (see figure 8.2 and Kunkel et al. (2014)). This data structure requires few bits per neuron in the network to signal presence or absence of targets for every possible source. For each source with local targets it incurs additional overhead besides the actual connection objects. For a fixed number of connections per neuron and a large number of MPI processes, a neuron typically has none or very few targets on an arbitrarily selected process (figure 8.1, cf. Kunkel et al. (2014)). Nevertheless, the sparse table occupies a major portion of the available memory in the post-petascale regime (figure 8.2, dark orange area, cf. Figure 2 in Kunkel et al. (2014)) as its memory consumption grows proportional to the total number of neurons in the network. This overhead limits the scalability of the simulation kernel. In addition, the size of the MPI receive buffers grows proportional to the total number of MPI processes due to the `MPI_Allgather` communication scheme (see figure 8.3A). The growth of spike buffers requires not only an increasing amount of memory (figure 8.2, gray area) but it also has an impact on simulation speed. For large-scale simulations over thousands of processes, a significant amount of time is spent on skipping spikes in MPI receive buffers from sources that do not have any local target - a contribution to the wall-clock time that grows linearly with the number of MPI processes (Schenck et al., 2014; Kunkel and Schenck, 2017a). To better exploit current supercomputers and to ensure scalability of the simulation code beyond the petascale regime, all data structures and runtime contributions that scale either proportional to the number of MPI processes or to the total number of neurons in the network need to be removed: Every process should only store and receive information relevant to local nodes.

Here we propose a solution that employs `MPI_Alltoall` (see figure 8.3B) to communicate spikes in a directed fashion, combined with a two-tier connection infrastructure for spike routing that consists of a presynaptic part, located on the MPI process of the sending neuron and a postsynaptic part, located on the MPI process of the receiving neuron. The connection infrastructure is constructed in two phases: First only the postsynaptic part including the actual connection objects is created and then, prior to simulation, the presynaptic infrastructure is constructed based on the postsynaptic connection information. In the following sections we discuss this new connection infrastructure and its instantiation in detail.

8.3.1 Two-tier connection infrastructure

Postsynaptic infrastructure

We first focus on the postsynaptic part of the connection infrastructure. On each process, we maintain two identically structured three dimensional vectors, the first storing connections that have process-local targets and the second their corresponding sources (figure 8.4A, top). Upon creation of a connection, the actual connection object and the source GID are pushed-back into the innermost dimension of these

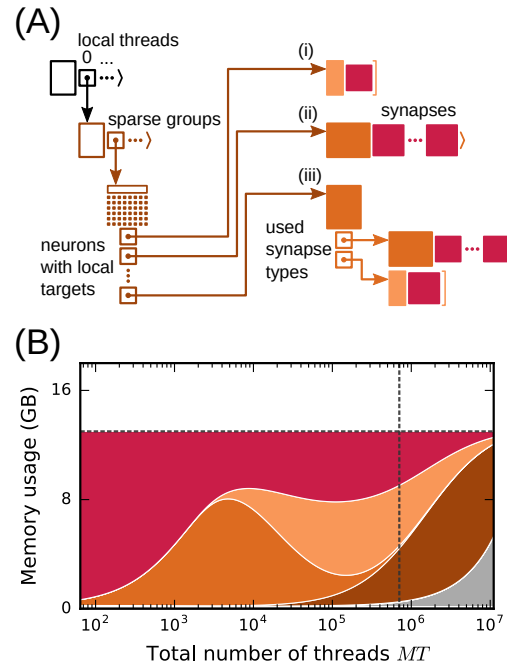


FIGURE 8.2: Connection infrastructure and memory usage for state-of-the-art purely postsynaptic storage of connections. (A) Connection infrastructure of the previous NEST kernel ('4g'). Each MPI process maintains a vector of pointers (top left; black) to thread-specific sparse tables (bottom left; dark orange), which hold connectivity information for all neurons in the network with local targets and enable efficient access to the corresponding connection objects. Each sparse table consists of sparse groups, where each group is responsible for 48 neurons: One bit per neuron indicates whether the neuron has local targets or not (tiny dark orange squares). If a neuron has local targets, the sparse group stores a pointer to a connector, which holds the local target synapses (pink filled squares). Depending on the number of synapses and the number of synapse types, the connector can hold (i) a small, fixed number of synapses of a single type (top right; light orange), (ii) a variable number of synapses of a single type (middle right; medium orange), or (iii) a connector of type (i) or (ii) (bottom right; medium orange) for each synapse type that is in use (bottom right; medium orange). (B) Estimated memory usage per compute node by different data structures of the 4g kernel as a function of the total number of threads for a maximum-filling scaling with 1 MPI process, 8 threads and 13GB of main memory per compute node. From top to bottom: synapses (pink), fixed-size connectors (light orange; case (i) in panel A), variable-size connectors and heterogeneous connectors (medium orange; case (ii) and (iii) in panel A, respectively), sparse tables (dark orange) and MPI buffers (gray). The dashed horizontal line marks the maximum memory available per compute node; the dashed vertical line marks the maximum number of MPI processes on the K computer.

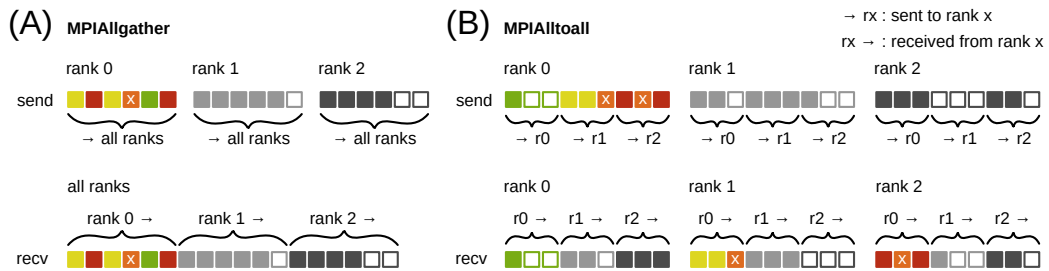


FIGURE 8.3: **Communication of receiver-selective data using MPI_Allgather and MPI_Alltoall.** The panels illustrate send and receive buffers for the example of an MPI communication that involves three ranks. Squares represent single buffer entries. Both collective MPI calls require homogeneous data types and equal send and receive buffer sizes for all ranks, which can entail sending empty buffer entries (unfilled squares). For the data that is sent by rank 0, colors indicate whether the data is required only by rank 0 (green), rank 1 (yellow), rank 2 (red), or both rank 1 and 2 (orange). For clarity, desired destinations for data that is sent by rank 1 and 2 are not indicated. **(A)** MPI_Allgather: All ranks receive the complete send buffer from all ranks, which can include unneeded data (e.g. rank 1 and 2 both receive the required orange entry but they also receive the unnecessary green entry). The receive buffer is a concatenation of all send buffers and the receive buffer size hence scales with the total number of ranks taking part in the communication. **(B)** MPI_Alltoall: Send buffers consist of equally sized sections that are destined for different receiving ranks, which allows each rank to define the data to be transmitted to any particular rank ; for example, rank 0 sends the yellow entries only to rank 1. Each rank has to send identically-sized buffer sections to each rank, which can entail sending empty buffer entries or even entirely empty buffer sections. Rank 2, for example, sends an empty buffer section to rank 1. To send specific data to multiple ranks, the sending rank needs to copy the data to the send-buffer sections of all intended receiving ranks, which leads to redundancy in the send buffer; rank 0, for example, sends the orange entry “x” to both, rank 1 and 2. The size of the receive buffers is identical to the size of the send buffers and independent of the number of ranks participating in the communication.

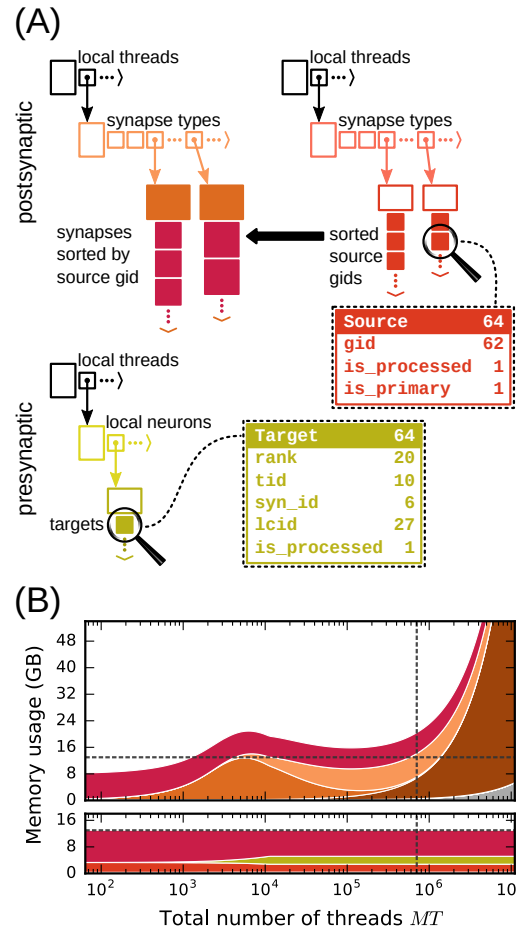


FIGURE 8.4: Two-tier connection infrastructure and its memory usage compared to purely postsynaptic storage of connection information. (A) Two-tier connection infrastructure of the new NEST kernel ('5g'). Top: The receiver side of the connection infrastructure consists of two identically structured parts: the connection table (top left) and the source table (top right). Connection table: Each MPI process maintains a vector of pointers (black) to thread-specific vectors of pointers to variable-sized containers for every synapse type. If a synapse type is in use, the corresponding container (orange filled rectangle) stores all thread-local synapses of this type (pink filled squares) in a vector. Synapse types can differ in memory consumption per object, indicated by different sizes. In each container, synapse objects are sorted by GIDs of the presynaptic neurons. Source table: Source objects (red filled squares) are stored in a three-dimensional vector, with a one-to-one relation between each source object and the synapse object in the same position in the connection table. Sources contain the GIDs of the presynaptic neurons and two markers (*Source* bit fields shown in dashed-line rectangle). Sender side of the connection infrastructure (bottom): Each MPI process maintains a target table, which is a vector of pointers (black) to thread-specific vectors of vectors that store the *Target* objects (green filled squares) for every thread-local neuron. The *Target* objects contain the locations of the targets, in terms of the MPI ranks and the three-tuple (*tid*, *syn_id*, *lcid*) that identifies the synapses on the target side in the corresponding connection table (*Target* bit fields shown in dashed line rectangle). (B) Estimated memory usage per compute node by different data structures of the 4g (top) and 5g (bottom) kernel as a function of the total number of threads for a maximum-filling scaling of the 5g kernel with 1 MPI process, 8 threads and 13GB of main memory per compute node. Color code for the 4g kernel as in figure 8.2. Color code for the 5g kernel from top to bottom: synapses (pink), targets (green) and sources (red). Memory usage of MPI buffers is not visible due to the small buffer size. The dashed horizontal line marks the maximum memory available per compute node; the dashed vertical line marks the maximum number of MPI processes on the K computer.

vectors indexed by the thread of the target neuron (first dimension) and type of the connection ('synapse type', second dimension). A single connection object contains all parameters and status variables of a synapse as well as a function to obtain a pointer to the target neuron. In these three dimensional structures, a three-tuple consisting of three integers describing local thread id, synapse id and local connection id, uniquely identifies a connection object on a specific MPI rank and can hence be used in a target-based AER scheme. In contrast to the previous generation kernel, connections are no longer separated according to source GID (cf. figure 8.2), thus avoiding overhead per potential source. The size of the data structure is therefore independent of the total number of neurons in the network. The removal of the separation by source also allows us to allocate the majority of the memory required to store all connection objects as a single chunk. To improve the size to capacity ratio of the used STL vectors, we use a resizing strategy that does not double the capacity each time the maximal capacity is reached, but allow the growth factor to be adjusted at compile time, with a default that increases the size of the vector by 50%. The additional storage of the GIDs of all source neurons in the second structure is required for the construction of the presynaptic part of the connection infrastructure (see section 8.3.1). This information also allows the user to query connectivity without requiring extensive MPI communication among all processes. Two additional flags are required within the source objects for construction of the presynaptic part of the connection infrastructure. They are implemented as bit fields to reduce the memory footprint (figure 8.4).

Presynaptic infrastructure

We now turn to the presynaptic part of the connection infrastructure required to implement directed communication of spikes. To determine the target processes to which a spike needs to be delivered, we need information about the locations of all postsynaptic targets on the process of the sending neuron. The presynaptic infrastructure consists of a single three dimensional vector which stores the location of all targets from outgoing connections of local neurons (figure 8.4A, bottom). The global location of a connection is uniquely given by the rank on which the target neuron is located in combination with the three-tuple for locating the connection in the postsynaptic structure on the target rank, as described in the previous section. To reduce the memory usage of this additional part of connection infrastructure, we have combined this location information in each object via bitmasks to fit into 8B, which strictly limits the maximal values of the individual fields (cf. figure 8.4A, bottom). The objects containing the location of all targets are stored according to the thread of the source (first dimension) and the thread-local id of the source (second dimension).

Construction of presynaptic connection infrastructure

We split network construction into two phases. The first phase stores connections and source GIDs in the postsynaptic connection infrastructure on the processes of the corresponding target neurons. The second phase, initiated by the simulation kernel just before the simulation starts, constructs the presynaptic infrastructure on all processes (see figure 8.5 for an example). All processes containing neurons with outgoing connections require information about the location of the corresponding targets. This information is communicated using `MPI_Alltoall` such that each

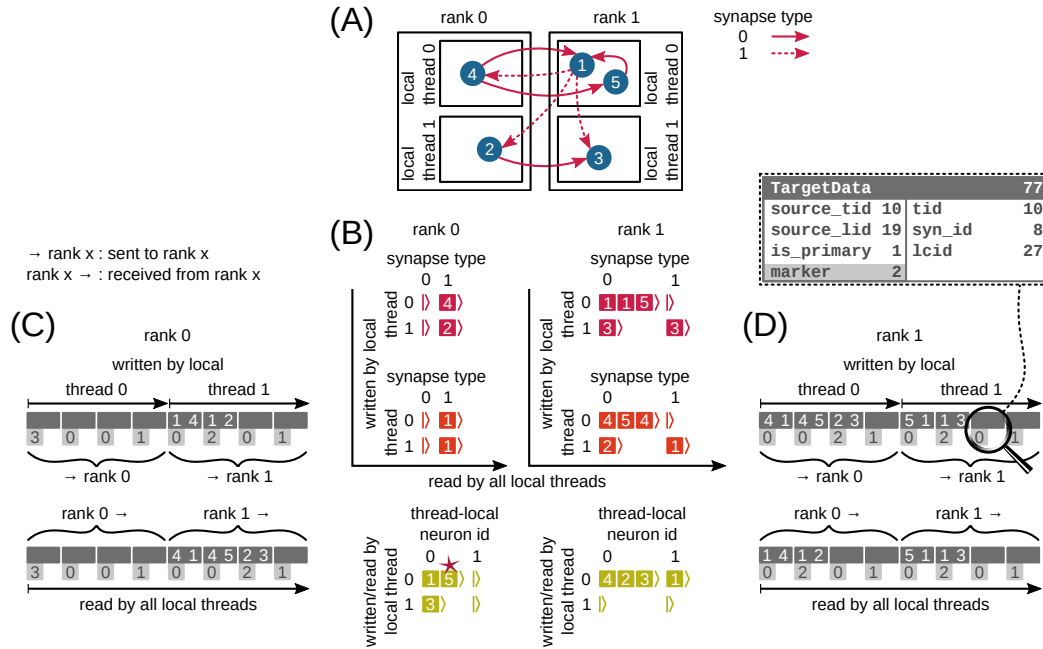


FIGURE 8.5: **Communication of connectivity data from postsynaptic to presynaptic side for the two-tier connection infrastructure.** Example network of 5 neurons (A) with global identifiers (GIDs) 1 to 5 (blue filled circles with white numbers) that are connected via two different types of synapses (pink arrows); for simplicity, the two types have synapse-type index 0 and 1 (solid and dashed arrows, respectively). Neurons are distributed across 2 MPI processes (outer rectangles) and 2 threads per process (inner rectangles); 4 threads in total. Synapses are hosted by the threads of their postsynaptic neurons. (B) From top to bottom: connection table, source table, and target table of the example network in panel A on rank 0 (left) and rank 1 (right). Color code as in figure 8.4A: Synapses, sources, and targets shown as pink, red, and green filled squares, respectively, where white numbers indicate target GIDs, source GIDs, and target GIDs again, respectively. The pink star indicates redundant connection information that is absent in the optimization for small-scale simulations (cf. section 8.3.3). All tables are three-dimensional vectors: outermost vectors for threads (vertical axes), middle vectors for synapse types or local neurons (horizontal axes), innermost vectors that hold the individual objects indicated by chevrons. When two neurons are connected, the thread of the postsynaptic neuron adds the new synapse to the connection table and a corresponding *Source* entry to the source table. Connectivity data needs to be communicated to the presynaptic side at the beginning of the simulation in order to construct the target table. (C, D) MPI send buffer (top) and receive buffer (bottom) that contain the *TargetData* of the example network, for rank 0 and rank 1, respectively; *TargetData* bit field shown in dashed line rectangle. Top rows (dark gray): Each field contains zero or two entries, which indicate the (source GID, target GID)-tuple. Bottom rows (light gray): Flags in each *TargetData* used for communication of status values among all processes (0: default, 1: no more data to send, 2: end of valid data in section, 3: skip this section).

rank receives only data about targets of its process-local neurons. While constructing the MPI send buffers on the postsynaptic side, each local thread is responsible for gathering the relevant information from the postsynaptic data structures for a range of source ranks. This design choice removes the strict separation of data structures by threads used in previous kernel versions. To construct the send buffer, each thread reads through all sources, determines the source rank for each connection based on the source GID and the round-robin distribution of neurons, and, if the source rank falls in its responsibility, creates an entry in the corresponding part of the MPI buffer (see figure 8.3B for an illustration of how different parts of the MPI buffers are distributed across ranks by `MPI_Alltoall`). A single entry in the MPI buffers comprises the thread and local id of the source neuron, the rank of the target neuron and the three-tuple identifying the corresponding connection in the postsynaptic structure. After the buffers have been exchanged via `MPI_Alltoall`, all threads read through the MPI receive buffers, each considering only entries relevant for thread-local neurons. From the relevant entries, the rank of the target neuron and the three-tuple are inserted into the presynaptic infrastructure. Prior to filling the send buffer, we use a collective MPI call to determine the maximal number of buffer entries that need to be communicated between any two ranks. This number is used to determine the required size of the MPI buffers in order to communicate all connection information in a single MPI call. The maximal size of the MPI buffers can be limited to avoid temporary peaks in memory consumption. In case this size is smaller than the required size, additional communication rounds are initiated until all connection information is communicated among all ranks. The current maximal MPI buffer size for communicating connection information is 128MB, which balances memory usage and number of communication rounds for our benchmark model. Under different circumstances, users might, however, want to reduce the number of communication rounds at the expense of larger memory usage or vice versa.

8.3.2 Communication of spikes

We now discuss the communication of spikes from the presynaptic to the postsynaptic ranks, making use of the two-tier connection infrastructure introduced above. Morrison et al. (2005) observe that the dynamics of neurons are decoupled from each other for the period of the minimal delay in the network. The authors exploit this insight by introducing a communication scheme that uses collective MPI calls for exchanging spikes at regular intervals determined by the minimal delay, instead of the typically much smaller simulation resolution. Here we rely on identical causality constraints to disentangle communication interval and simulation resolution, but modify the communication algorithm to make use of directed communication via `MPI_Alltoall`. During a communication interval, spikes are buffered in a four dimensional vector (figure 8.6) and only transferred to the MPI send buffers at the end of the interval, right before the MPI communication takes place. In the four dimensional data structure, the first dimension corresponds to the thread of the source neuron, the second dimension corresponds to the thread responsible for writing the respective entries to the MPI buffer (see below), and the third dimension corresponds to the relative simulation step within the current communication interval at which the spike occurred ('lag'). In a similar way as in the collocation of the send buffer during communication of connection information, each thread is assigned a range of receive ranks for which it is responsible, corresponding to particular sections in the MPI send buffer (see figure 8.3, cf. section 8.3.1). By splitting the MPI

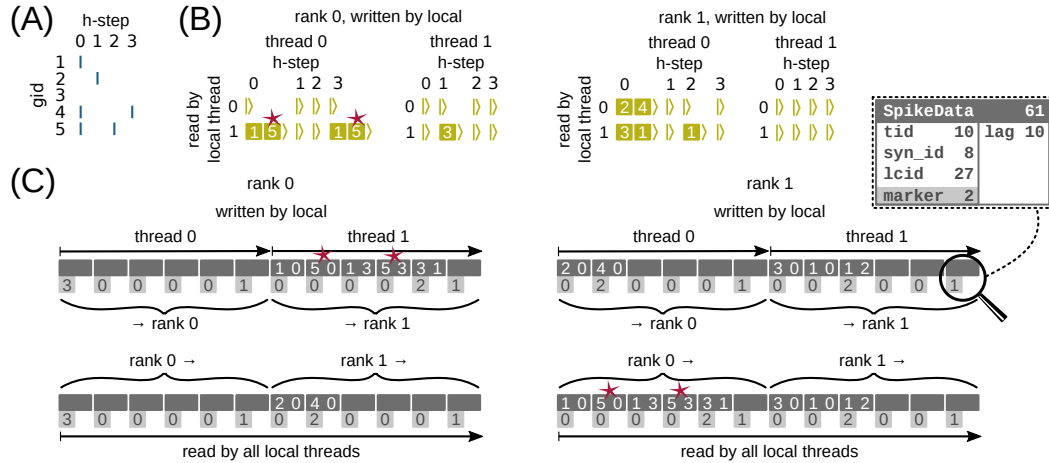


FIGURE 8.6: **Communication of spike data using MPI_Alltoall** for the example network in figure 8.5A and an example activity (A), where a communication step consists of four neuronal update steps (*h*-steps); spikes are shown as blue bars. (B) Spike register, which temporarily buffers spikes before they are collocated in the communication buffers on rank 0 (left) and rank 1 (right). Numbers indicate target GIDs. Pink stars indicate redundant information, absent in the optimization for small-scale simulations (cf. section 8.3.3). (C) MPI send buffers (top) and receive buffers (bottom) for rank 0 (left) and rank 1 (right) that contain *SpikeData* for the example activity. *SpikeData* bit fields shown in dashed rectangle. Top rows: each field contains zero or two entries, which indicate the (target GID, lag)-tuple. Bottom rows: flags in each *SpikeData* used for communication of status values among all processes (0: default, 1: no more data to send, 2: end of valid data in section, 3: skip this section).

send buffer in this fashion, buffers can be collocated in parallel, without requiring additional checks to avoid that threads are writing to the same memory address. When a neuron spikes, the corresponding thread retrieves the locations of all its outgoing connections from the presynaptic infrastructure (section 8.3.1). For each entry the thread appends a copy of each target at a specific position in the four dimensional spike buffer, determined by the thread of the source neuron (first dimension), the id of the thread that will later create the corresponding entry in the MPI buffer (second dimension), and the current lag (fourth dimension). The second dimension is introduced in order to read the spike buffer in parallel at the end of the communication interval: Each thread only processes its (private) share of the spike buffer and creates entries in the MPI buffer accordingly. An entry comprises the three-tuple identifying a connection on the receiving rank through which the spike should be delivered to the target neuron, while the receiving rank is implicitly encoded by the entry's position in the MPI send buffer. After the spike buffer has been completely processed or in the case that only spikes are left that cannot anymore be accommodated in the MPI buffer, buffers are exchanged among all compute nodes via `MPI_Alltoall`. Directly after the MPI communication, all threads read the process-local MPI receive buffers and deliver spikes only via thread-local connections. Several of such communication rounds may be required until all MPI processes have completely exhausted their spike buffers. Each additional communication round at the end of a single communication interval increases the size of the MPI buffer up to a certain, user adjustable, maximal size. This dynamic resizing allows the MPI buffers to maintain the minimal required size, reducing the total number of required communication rounds. After all ranks have processed their spike buffers, the simulation resumes with updating the nodes for the next interval.

8.3.3 Adaptations for the pre-petascale regime

The two-tier infrastructure introduced above is designed for the regime, where each neuron has either no or only few targets on each thread ($K \ll MT$; “sparse limit”, Kunkel et al. (2014)), motivated by the sub-optimal memory usage of the previous technology in this limit (figure 8.2). However, in the pre-petascale regime, neurons typically have many targets per thread ($K \gg MT$, see figure 8.1). This results in performance degradation of the new simulation technology compared to the previous technology, due to: (i) reduced memory locality in the postsynaptic infrastructure as connections are not separated and stored sequentially by source, leading to hardly predictable random memory access during spike delivery and (ii) redundancy in communication of spikes as each source sends an individual spike to each of its targets and hence possibly multiple spike to each thread (red stars in figure 8.6). In the extreme case of a single MPI process with a single thread, each neuron generates K spikes prior to MPI communication, instead of just a single packet that is multiplied during delivery on the postsynaptic side as it was the case for the previous technology. On these grounds Morrison et al. (2005) argue that for a small number of processes, a sender-based AER scheme is most efficient.

Here we solve both shortcomings by sorting connections in the postsynaptic infrastructure by their corresponding source GIDs (figure 8.4). A spike delivered to the first connection originating from a specific source can then (locally) be passed to all subsequent connections of the same source, leading to linear memory access patterns. The source neuron hence only needs to send a single spike to each thread on which it has targets, postponing some of the spike duplication to the postsynaptic delivery. In addition to solving the above issues, this reduces the memory usage of the presynaptic infrastructure in the regime where neurons have multiple targets per thread, as a single neuron only needs to store the location of MT targets instead of K targets. During construction of the presynaptic connection infrastructure each thread only communicates the location of the first connection for each combination of source and synapse type. Due to the organization of the data structure sources can only be sorted per synapse type. Therefore, this optimization is only effective if a minimal number of different synapse types necessary for a particular simulation is used. For example, in the NEST simulation software the `CopyModel` mechanism is a convenient way to create a synapse type that differs from the original by the values of default parameters. This makes the description of the network model in the simulation language more compact and readable. The consequence, however, is that the above optimization does not work as efficiently as for a model description where varying synapse parameters are explicitly set for different synapse instances of the same type. As the number of MPI processes increases the benefit of sorting connections by source diminishes. Not using the optimization saves time during network construction for large-scale simulations by avoiding the then superfluous sorting step (section 8.3.5).

8.3.4 Memory usage of the new simulation kernel

The memory-usage model that is introduced in section 8.2.5 exposes the differences between the previous (4g) and the new (5g) simulation kernel in total memory usage per MPI process of our reference network (section 8.2.1). Here we consider a maximum-filling scenario (section 8.2.2) for the 5g kernel using 13GB per MPI process. In the previous kernel, the sparse tables and MPI buffers consume memory proportionally to the total number of neurons in the network ultimately limiting

the scalability of the simulation code (figure 8.2). Removing the sparse table and the constant overhead for all sources with local targets by introducing a two-tier connection infrastructure significantly reduces the memory usage in the petascale regime and beyond ($MT > 10^5$, see figure 8.4). Additionally, replacing `MPI_Allgather` with `MPI_Alltoall` reduces the size of MPI receive buffers by eliminating their growth with the total number of processes. These changes result in perfect scaling of the postsynaptic data structures with respect to memory usage. The new technology allows for simulations of larger networks as the previous kernel using the same computational resources in the regime of more than a few hundred MPI processes (figure 8.4). For more than 10^3 MPI processes, also the presynaptic infrastructure consumes constant memory as its size is proportional to the average out-degree of the neurons ($\sim K$). In the regime up to 10^3 MPI processes, we achieve decreased memory usage of the presynaptic infrastructure due to the optimizations described in section 8.3.3. In this regime, each neuron typically has many targets on each thread, such that the size of the presynaptic infrastructure is proportional to the total number of threads ($\sim MT$). Below $\sim 10^2$ MPI processes, the previous technology consumes less memory as it does not require storage of the source GID for every connection. In addition, the size of STDP synapse objects has increased, since each STDP synapse now needs to store the time of the last spike of its source, which previously only needed be stored once per source neuron on each thread. To reduce the total memory consumption, the stored source GIDs can optionally be removed while constructing the presynaptic data structures. This however implies that after the presynaptic infrastructure has been created, network connectivity can neither be changed nor queried any more.

8.3.5 Performance of new simulation kernel

To investigate the performance of the new technology we next carry out scaling experiments using the archetypal network model outlined in section 8.2.1. In the following, we use the terms ‘build time’, ‘presim time’, and ‘sim time’ to refer to the measured wall-clock times of three successive phases of the benchmark-network simulations. The build time is the wall-clock time required to construct all nodes and the postsynaptic part of the connection infrastructure. After the build phase a very short pre-simulation of $\mathcal{O}(10\text{ ms})$ biological time is followed by the actual simulation of $\mathcal{O}(1\text{ s})$ biological time; the measured wall-clock times are referred to as presim time and sim time, respectively. The presim time accounts for the construction of the presynaptic part of the connection infrastructure in 5g (see section 8.3.1), but also for the initial resizing of MPI buffers in both 4g and 5g. Besides, initial transients of the network dynamics subside during the pre-simulation. In the following we refer to the 5g kernel version that uses the adaptations for the pre-petascale regime described in section 8.3.3 as ‘5g-sort’ and to the version without these adaptations as ‘5g-nosort’; collectively we refer to both versions as ‘5g’.

Weak scaling

In weak scaling benchmarks, the problem size per compute node, or equivalently per MPI process, is fixed while the total number of compute nodes increases (section 8.2.2). In this scenario a perfectly scalable simulation kernel shows constant run time for all phases and constant memory usage per compute node. We simulate our benchmark-network model (section 8.2.1) on JUQUEEN with $N_M = 18,000$ neurons per compute node and $K = 11,250$ incoming synapses per neuron while

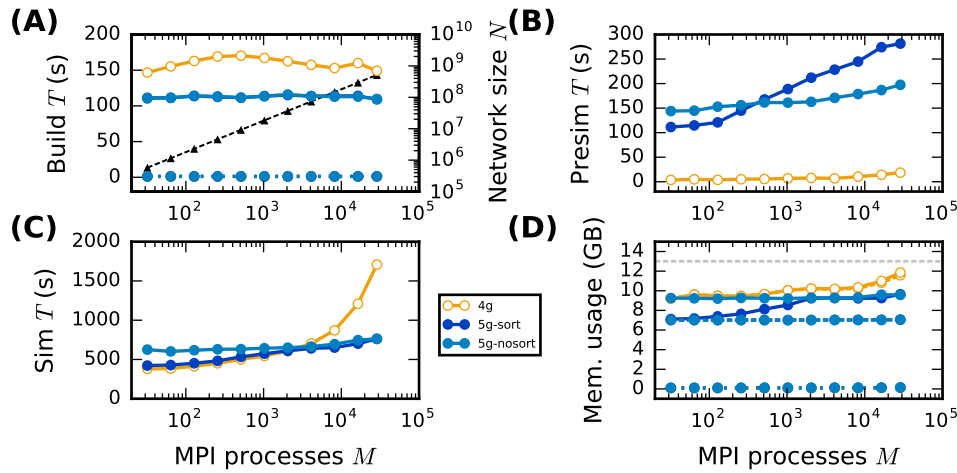


FIGURE 8.7: **Weak scaling of an archetypal neuronal network simulation on a petascale computer.** Runtime and memory usage per compute node for an increasing number of MPI processes $M \in \{32; 64; 128; 256; 512; 1024; 2048; 4096; 8192; 16,384; 28,672\}$ with one MPI process and $T = 8$ threads per compute node on JUQUEEN. Each compute node hosts $N_M = 18,000$ neurons with $K = 11,250$ incoming synapses per neuron. Network dynamics simulated for 1s of biological real time. Color code in all panels: 4g (orange), 5g without optimizations for pre-petascale ('5g-nosort', light blue), and 5g with optimizations ('5g-sort', dark blue). **(A)** Build time of nodes (dashed lines) and connections (solid lines); 5g-sort data covered by 5g-nosort data. Black triangles show the total network size N (right vertical axis). **(B)** Presim time. **(C)** Sim time. 4g data partly covered by 5g-sort data. 5g-sort data partly covered by 5g-nosort data. **(D)** Cumulative memory usage for a single MPI process after construction of neurons (dotted lines; $< 140\text{MB}$), after construction of connections (dashed lines), and after simulation including pre-simulation (solid lines). 5g-sort data partly covered by 5g-nosort data. Horizontal gray dotted line indicates the maximum memory available on a single compute node of JUQUEEN.

build time, presim time, sim time, and memory usage per compute node assess the performance of the 4g, 5g-sort, and 5g-nosort variants. The build time shows almost perfect scaling for both 4g and 5g, as construction of the postsynaptic part of the connection data structures is completely parallel and does not involve any communication among compute nodes (figure 8.7A). While the build time for nodes is negligible for both technologies, construction of connections is about 25% to 30% faster in the new kernel. This is due to a more efficient memory allocation for synapse objects in 5g: The major part of the memory that is required to store all local connections of a specific type is allocated en bloc (section 8.3.1). Ippen et al. (2017) discuss the intricacies of memory allocation during network construction in the previous kernel. Note that for simulations using the previous kernel, we preload jemalloc (Evans, 2011) to replace the default allocator.

In the following discussion we focus on the comparison of 4g and 5g-sort, and afterwards consider the differences to 5g-nosort. After construction of the postsynaptic data structures in the new kernel, the connection information needs to be communicated to the presynaptic side, leading to a significant increase in presim time compared to the previous kernel (figure 8.7B). For a small number of MPI processes ($\lesssim 128$) the presim time is evenly spent on sorting connections by the respective sources and on collocating connection information to the MPI send buffers. To collocate the MPI buffers, all threads in a single process need to read through the

whole postsynaptic data structures and therefore the performance gain from multiple threads is small in this phase. For a larger number of processes the contribution from collocating buffers increases significantly over the investigated range of number of nodes, dominating the presim time in the regime of 10^4 MPI processes and leading to a significant increase in presim time over the investigated range of MPI processes. On the one hand, the increase of work load is due to the growing amount of connectivity data that needs to be exchanged between processes, because targets of individual neurons are distributed over more and more ranks (cf. section 8.3.3). On the other hand, the increase is due to suboptimal usage of the available space in the MPI buffers caused by the random out-degree of nodes. The presim time, however, is a constant contribution to the total wall-clock time and therefore its relative impact decreases for longer simulation times. The sim time is approximately equal in the two kernel versions up to about 2048 compute nodes. For larger numbers of MPI processes, the sim time in 4g increases approximately linearly with the number of MPI processes due to the linearly increasing size of the MPI receive buffers. Since the receive buffer size stays approximately constant in 5g, sim time increases only slightly, leading to much better scaling behavior and a decrease in sim time by more than 55% for full JUQUEEN simulations (figure 8.7C). The main contribution to the sim time are the delivery of spikes from the MPI buffers via the respective connections to the target neurons, including weight updates of plastic synapses, followed by the time spent on propagating the neuronal dynamics. Over the investigated range of MPI processes, communication time takes up only a small part of the total simulation time (figure 8.15). The relative contribution of communication, however, depends on the minimal delay in the network, which determines the communication interval (here $d_{\min} = 1.5$ ms). Communication can become a relevant factor in simulations with small delays on the same order of magnitude as the simulation resolution ($d_{\min} \sim 0.1$ ms). The memory usage per compute node is consistently smaller in the new kernel than with previous technology; in particular it is approximately constant above ~ 2048 processes as predicted by the memory model (section 8.3.4), demonstrating its perfect scaling behavior. In this regime, most memory is allocated for the individual connection objects, followed by contributions from the postsynaptic infrastructure responsible for storing the source of each connection and a significant contribution from the presynaptic infrastructure storing the targets of neurons. The slight increase in memory usage for more than 10^4 processes is most likely caused by an increased memory usage of the MPI library. For a smaller number of MPI processes, the optimizations for small-scale simulations reduce the size of the presynaptic data structures significantly and hence the memory usage of the new kernel by about 20% compared to the case without optimizations (figure 8.7, blue vs. light-blue). In addition to simulations on JUQUEEN, we perform weak-scaling benchmarks on the K computer. The results are very similar to the data obtained from JUQUEEN (figure 8.8), demonstrating the excellent scalability of the simulation kernel to almost 10^5 MPI processes. The small increase in memory consumption for the largest simulations ($M = 82,944$) can be traced back to the increased memory usage of the MPI library.

Without optimizations for small-scale simulations (5g-nosort), the presim time increases for a small number of MPI processes ($M < 10^3$) compared to 5g-sort due to an increased amount of connection information that needs to be communicated among ranks. However, as the number of MPI processes increases, a single neuron has a decreasing number of targets on each process, diminishing the advantage of sorting connections by source. In this regime, we save time by not sorting connections without incurring a penalty, leading to 5g-nosort having a smaller presim time

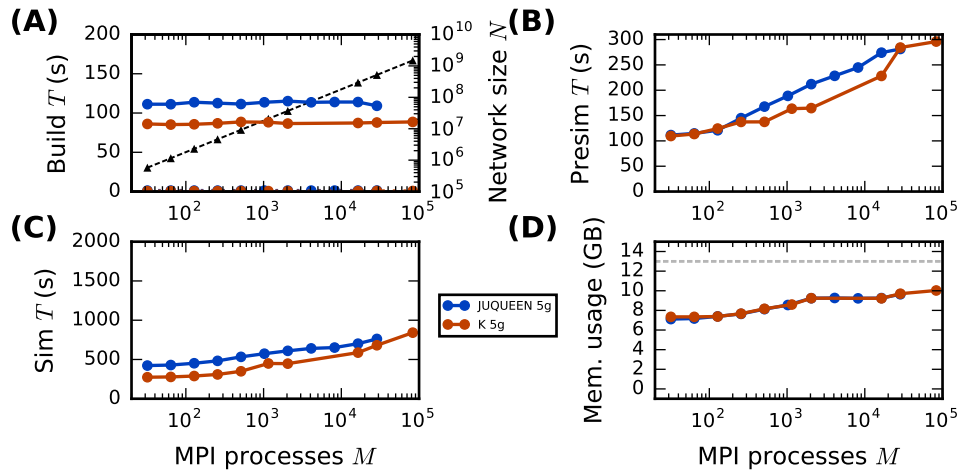


FIGURE 8.8: **Comparison of weak scaling on two petascale systems, JUQUEEN and K computer.** Color code in all panels: 5g-sort on JUQUEEN (dark blue), 5g-sort on the K computer (dark red). See figure 8.7 for details. Number of MPI processes: $M \in \{32; 64; 128; 256; 512; 1152; 2048; 16,384; 28,672; 82,944\}$ with a single MPI process per compute node and $T = 8$ threads per process.

than 5g-sort from about 1024 MPI processes on. Also the sim time for 5g-nosort is increased in the regime of few MPI processes, due to a larger amount of spikes that need to be communicated: one for each target per thread, instead of just one for the first target per thread. This disadvantage diminishes with increasing number of MPI processes, and 5g-sort and 5g-nosort exhibit similar sim times from about 2048 MPI processes on. In contrast to 5g-sort, memory usage for 5g-nosort is constant since the size of the presynaptic data structures are independent of the number of MPI processes also for $MT < K$.

Strong scaling

Increasing the number of MPI processes in a strong scaling scenario (section 8.2.2) shows perfect scaling of the build time for both the old and the new simulation kernel since construction of nodes and the postsynaptic part of the connection infrastructure are entirely parallel and do not require any communication (figure C.5). Also the presim time shows almost perfect strong scaling for the new kernel. For the previous kernel, the sim time initially scales very well, but saturates between 2048 and 4096 MPI processes; it even increases for larger numbers (figure 8.9), most likely due to the increasing size of MPI receive buffers as reading of the buffers introduces significant overhead during spike delivery. Using the new kernel, the sim time scales well up to 8192 processes where it saturates, simulating 1s of biological real time in 16s of wall-clock time for a network with more than 10^6 neurons and almost 10^{10} plastic synapses (figure 8.9). Deviations from perfect scaling can mainly be traced back to MPI communication, which puts a strict lower bound on potential optimization of the simulation time. The memory consumption of the new kernel before construction of the presynaptic infrastructure scales perfectly with increasing number of MPI processes. However, as targets of neurons become distributed across more and more MPI processes, the size of the presynaptic data structures increases, leading to slightly higher memory usage for a large number of MPI processes (figure C.5).

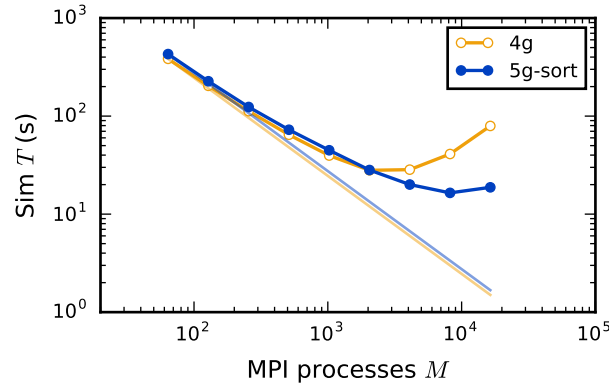


FIGURE 8.9: **Strong scaling of simulation time of an archetypal neuronal network simulation on a petascale computer.** Same color code, marker styles, and line styles as in figure 8.7. Straight solid lines indicate perfect scaling. $N = 1,152,000$ and $M \in \{64; 128; 256; 512; 1024; 2048; 4096; 8192; 16,384\}$ and $T = 8$ threads per process.

Maximum-filling scaling

The maximum-filling scaling scenario keeps the memory usage per compute node approximately constant and close to the maximal memory available. This exposes the limits of the new technology in terms of the maximal network size on a given hardware (section 8.2.2). Since without optimizations the memory usage per compute node stays constant, weak scaling and maximum-filling scaling are identical in that case. Here we consider the case with optimizations (5g-sort). The memory usage of the presynaptic connection infrastructure is negligible for small-scale simulations and increases as the targets of each neuron become more and more distributed across different compute nodes. The number of neurons that fit on a single compute node consequently decreases, up to about 2048 processes (figure 8.10). For a larger number of MPI processes, the number of neurons fitting on one compute node stays constant, allowing us to simulate about 730 million neurons connected with about 8 trillion synapses employing the entire JUQUEEN supercomputer. Similar to the weak scaling scenario, simulation time scales well across the investigated number of processes, increasing by less than 45% for an almost 1000 fold increase in network size. The memory model delivers a fairly accurate prediction of the actual memory usage of a simulation (figure 8.10, compare light gray and dark blue solid lines) with a deviation of about 5%. This mismatch is most likely caused by dynamically resizing containers in the connection data structures for which an accurate estimate of the actual memory usage is difficult.

Performance for small-scale simulations

Although the data structures and algorithms presented in this work are designed to improve the scalability of simulations on large-scale HPC systems, optimizations for small-scale to medium-scale systems (section 8.3.3) assure that the proposed solution does not impair the performance on laptops, workstations, and small clusters. A down-scaled version of the network model used throughout this study (section 8.2.1) assesses the performance of the new technology in light of the old (figure 8.11). The test system is a Lenovo X250 equipped with an Intel Core i7-5600U processor running at 2.6GHz with 8GB of main memory. Similar to the results on HPC systems, the build time of nodes and postsynaptic connection infrastructure decreases by 50% compared to the previous technology (figure 8.11). As before, the presim phase is,

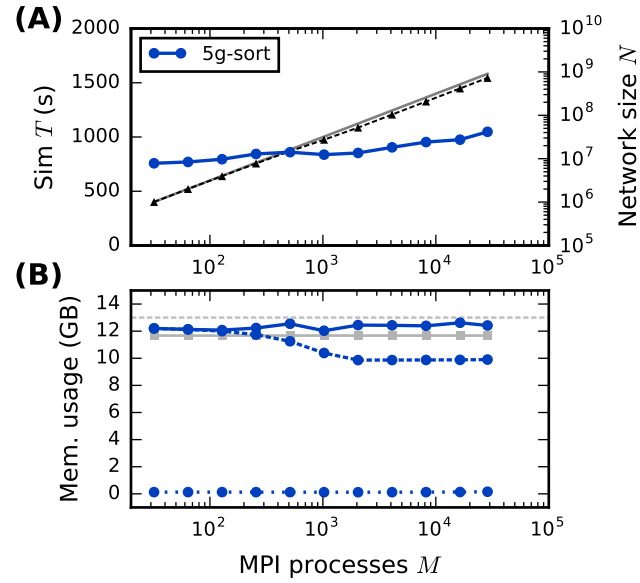


FIGURE 8.10: **Maximum-filling scaling of an archetypal neuronal network simulation on a petascale computer.** (A) Runtime and (B) memory usage per compute node for an increasing number of MPI processes $M \in \{32; 64; 128; 256; 512; 1024; 2048; 4096; 8192; 16,384; 28,672\}$ with one MPI process and $T = 8$ threads per compute node of JUQUEEN. Same color code, marker styles, and line styles as in figure 8.7. Black solid line in (A) indicates network size for perfect scaling. Light gray solid line with square markers in (B) is the prediction of the memory model.

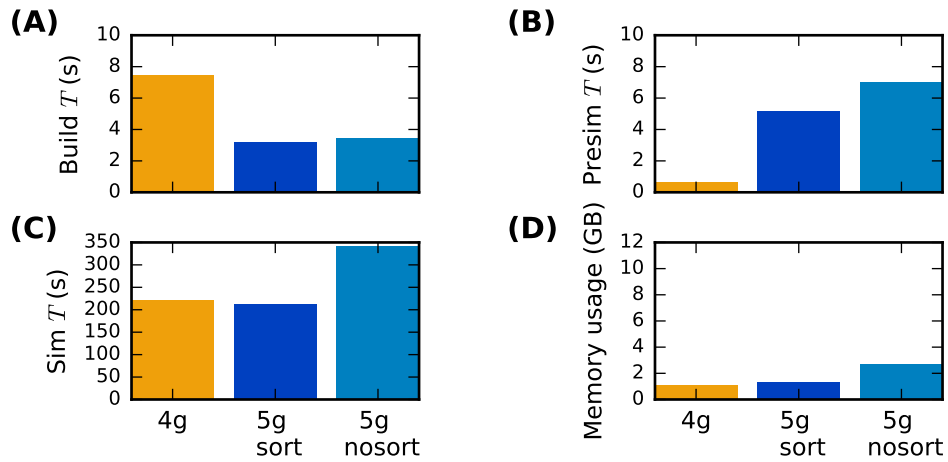


FIGURE 8.11: **Performance of small-scale simulations.** Comparison of (A) build time, (B) presim time, (C) sim time and (D) memory usage for a down-scaled typical network model on a laptop for 4g (orange), 5g-sort (dark blue) and 5g-nosort (light blue). $N = 11,255$ and $K = 3750$. Simulations run for 1s of biological real time.

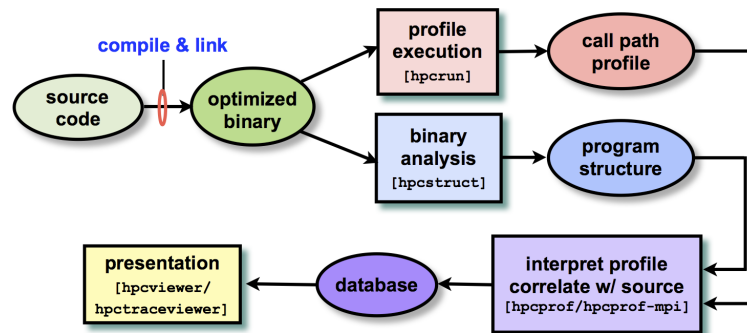


FIGURE 8.12: **HPCToolkit workflow.** Applications are compiled and linked as usual, with optimizations enabled. Running the application through `hpcrun` enables the collection of performance metrics, while `hpcstruct` recovers the program structure. These performance metrics are combined with the program structure to attribute for example timing measurements to specific parts of the code via `hpcprof`. Finally, one can visualize the results using `hpcviewer` or `hpctraceviewer`. Figure from <http://hpctoolkit.org/>.

however, much longer due to the required construction of the presynaptic part of the connection infrastructure. In total this leads to comparable network construction time for the previous and new technology (build time + presim time). The sim time is almost identical in 4g and 5g. This time is almost exclusively spent on delivering spikes to neurons and on updating plastic connections, both of which are of similar computational complexity in the two technologies. Without optimizations for small-scale simulations, the sim time increases by about 75%, explicitly demonstrating the necessity of these modifications. Memory usage is also almost identical for the two technologies in the presence of the optimizations. Otherwise, the presynaptic part of the connection infrastructure consumes a significant amount of additional memory.

8.4 Detailed profiling via HPCToolkit

Manually instrumenting the source code of a program to measure the time spent in specific parts of the software provides rough estimates of its performance characteristics and can be useful for guiding optimization efforts. And yet, it is not feasible to obtain very detailed information about the time spent in specific functions, in particular not if they are frequently called – such heavily used parts of the code, however, should be the main focus of optimizations. For example, manually inserting timing measurements in frequently called functions leads to a systematic bias, since the timing code itself has a small overhead. To address the issue of biased measurements, tools like the HPCToolkit (Adhianto et al., 2010) avoid changes in the original source code, and instead use sampling techniques. Such techniques can provide a very fine grained view of the relative contributions of different parts of the program to the total runtime. The basic concept is to send a signal repeatedly to the program being profiled, triggering the recording of the current context. Collecting many of these samples, reveals how much time was spent in each context relative to the total runtime. By combining this information with static code analysis, HPCToolkit allows researchers to directly map the measurements to specific parts of the source code (figure 8.12) and can hence reliably guide optimization efforts. In addition it is possible to record a timeline of contexts encountered, a so called program-execution trace, useful for example to detect load imbalance between MPI processes or threads.

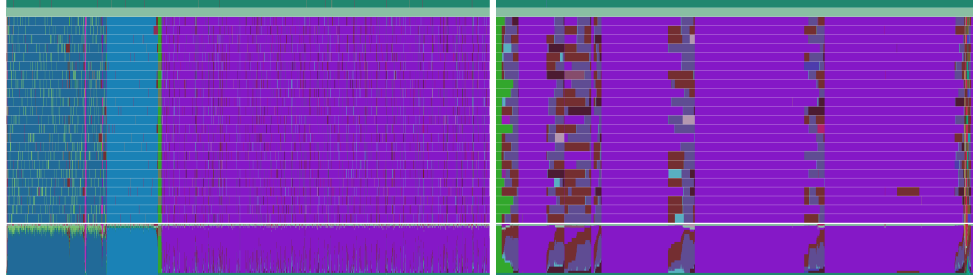


FIGURE 8.13: **Trace view of a simulation of a network model with the new simulation technology** resolved by threads (top) and summary view (bottom). All synapses are static. Left: Coarse view of the full simulation duration, with an approximate separation into build (dark blue), presim (light blue) and sim phase (purple). Right: Fine view just after entering the sim phase.

Here we use the HPCToolkit to profile two runs of the benchmark network model introduced in section 8.2.1, once with and once without plastic synapses and provide a preliminary analysis of the results. While the HPCToolkit supports profiling of large-scale applications running thousands of MPI processes, we use for this initial investigation a single MPI process with 24 threads. To reduce the computational load, the model is downscaled to 10^4 neurons, each receiving 2800 connections from other neurons in the network. The data was collected on a single compute node of a local cluster hosting two Xeon E5-2680 v3 processors with 128GB of main memory and revision 42ed811 of the NEST 5g development branch.

A trace of the program execution for the simulation without plastic synapses shows a clear separation of build, presim and sim phase (figure 8.13, left). In this coarse view, all threads seem to be engaged equally, and we do not observe any severe load imbalances. During the build phase (dark blue), the vertical pink line arises from all threads reserving memory for connection objects at approximately the same time. This can lead to memory-allocation bottlenecks for non-thread-aware allocators, which could be mitigated by preloading a modern allocator like jemalloc (Ippen et al., 2017). After the initial build phase, a significant amount of time is spent during the presim phase on sorting connections and the corresponding sources (light blue, cf. figure 8.15). At the end of the presim phase, the green vertical line indicates idling of a few threads, most likely caused by the single-threaded collocation of MPI send buffers for distribution of connection information (cf. section 8.3.1). In the sim phase, most time is spent on gathering and distributing spikes (purple). A detailed resolution of the sim phase (figure 8.13, right), reveals that in addition to the delivery of spikes from the MPI receive buffer (purple), a substantial amount of time is spent on generating and distributing events from Poisson generators (dark purple), and that threads spend quite some time idling (dark red). A more extended analysis is necessary to find and remove the source for this observed imbalance and possibly improve the performance of the Poisson generators.

Collapsing the time dimension allows us to attribute relative contributions of particular functions to the total runtime in terms of the `PAPI_TOT_CYC` counter (figure 8.14). In the following we use inclusive measures, which means that all measures include the runtime of the function and all functions called within the function, and we use the term time synonymous to PAPI cycles. In both models, with and without plasticity, most of the time is spent in the sim phase, followed by contributions from the build phase and the presim phase. In the network without plasticity most time during the propagation of network dynamics is spent on the delivery of

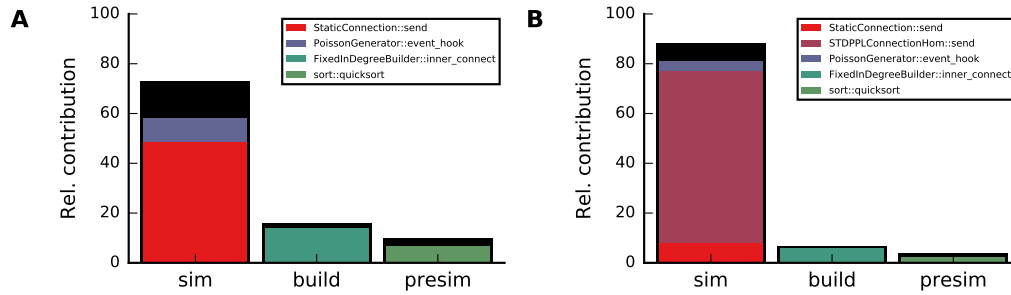


FIGURE 8.14: Relative contributions to the total runtime. **A:** Network model with static connections. **B:** Network model with plastic connections.

spikes via connections to the spike buffers of the respective neurons (figure 8.14, red). These spike buffers are necessary to accommodate spikes that carry a dendritic delay. This frequent, unpredictable access to spike buffers distributed anywhere in main memory violates the principle of locality, leading to suboptimal cache usage – the CPU spends many clock cycles on waiting for the relevant data. Reduction of this contribution most likely requires significant changes to the relevant data structures (Schenck, Zaytsev, and Morrison, 2014; Kunkel and Schenck, 2017b). Another significant contribution to the runtime arises from spike generation via Poisson sources (figure 8.14, dark blue). The storage of connections for devices separated from connections between neurons introduced in the new kernel (section C.2) can potentially reduce this contribution. During the build phase most time is spent in the inner loop of the specific connection builder employed (figure 8.14, dark green) which has just recently been optimized (Ippen et al., 2017). In the presim phase, connections are sorted by source to allow efficient delivery of spikes during the propagation of network dynamics (section 8.3.3). This requires in-place sorting of connection and source objects simultaneously, both stored in large vectors. Due to many identical keys (GIDs of source neurons), a three-way quicksort is a suitable choice (Sedgewick and Wayne, 2011) and it is questionable whether alternative sorting algorithms can significantly improve this. However, this contribution can potentially be further reduced by modifying connection algorithms to create partially-sorted arrays and adapting the sorting algorithm to deal with these efficiently. For the simulation of the network model with plastic synapses, the relative time spent in the sim phase is significantly increased and dominated by the spike delivery via plastic synapses. In particular, for the plasticity rule employed here, many calls to `std::exp` consume a large part of the total runtime. The large contribution to the wall-clock time makes this part of the code an attractive target for further optimizations. While these measurements concern one specific model of plasticity, calls to the base class of neurons during the update of plastic synapses take up a significant portion of the computation time and optimizing these could benefit a larger class of plasticity models. Suitable improvements need further investigation of the various plasticity rules typically employed by the neuroscience community.

In this section, we have used the sampling-based profiling tool HPCToolkit to obtain a more detailed analysis of various contributions to the overall runtime of a neuronal network simulation without modifying the original source code. In contrast to manually placed timers, such tools provide a detailed, but undistorted view of the relative contributions and hence facilitate the identification of optimization objectives. Even from this initial analysis, it is clear that to significantly accelerate single-process, multi-threaded simulations one needs to improve the delivery of

spikes to their target neurons. On the one hand this implies optimization of specific plasticity rules, and on the other hand also the improvement of the current data structures to optimize memory access patterns during spike delivery. While here we have focused on a single process with multiple threads, future investigations should consider multi-process, multi-threaded simulations. The metrics derived from the bare measurements will provide additional insights into scalability bottlenecks when simulations with varying numbers of processes or threads are performed.

8.5 Discussion

Well-tested high-performance simulation tools are a prerequisite for advancing neuroscientific modeling of large-scale networks at cellular resolution. The complex connectivity of biological neural networks poses a challenge for the numerical simulation of such systems. Previous technologies scale neuronal network simulations to the largest computing systems available today. The work that we present here explores the scalability of new data structures and communication schemes in terms of memory usage and runtime on post-petascale systems and proposes concrete implementations.

The investigation starts by identifying scalability bottlenecks in the previous simulation kernel (4g): In large-scale simulations, a prohibitively large fraction of the available memory on each compute node is taken up by the connection infrastructure and MPI buffers. Therefore, we subsequently design scalable alternative data structures that fully exploit the sparsity of large-scale networks distributed over tens of thousands of MPI processes. The goal of these technological improvements is not to reduce memory consumption per se, but to decouple memory consumption per MPI process from the total network size and the total number of MPI processes. In the new kernel (5g) we introduce a two-tier connection infrastructure that combines a postsynaptic part located on the same rank as the target neurons with additional presynaptic data structures located on the ranks of the sending neurons. The presynaptic structure allows us to introduce directed MPI communication via `MPI_Alltoall` and to remove non-essential information from the postsynaptic rank. This design choice significantly improves the scalability of the simulation kernel without sacrificing flexibility and generality. Through the introduction of the two-tier connection infrastructure and the direct communication via `MPI_Alltoall`, the new kernel achieves perfect scaling in terms of memory consumption, which prepares the simulation technology for next generations of large-scale HPC facilities. The new data structures and communication schemes enable the simulation of a network of $1.51 \cdot 10^9$ neurons and $16.8 \cdot 10^{12}$ synapses on the K computer (figure 8.8), in terms of connectivity slightly larger than the previously reported (Kunkel et al., 2014) simulation with $11.1 \cdot 10^{12}$ synapses. This simulation does not require the full main memory available on the K computer; the maximal network size could be increased by about 20% before running out of memory. Our work focuses on a decrease in memory usage for post-petascale systems - yet the new data structures and communication framework lead to significant performance gains for large-scale simulations on current supercomputers. Simulations employing the full supercomputer JUQUEEN now run up to 55% faster than previously, mainly due to the fixed size of MPI receive buffers. The novel technology profits from sending only the relevant spikes to each process: For more than 10^4 MPI processes, most of the spikes arriving via `MPI_Allgather` in the old scheme are not required locally,

but still need to be processed, leading to a large serial overhead during spike delivery (Schenck et al., 2014; Kunkel and Schenck, 2017a). The constant size of the MPI receive buffer independent of the number of MPI processes also significantly improves the strong scaling behavior. For $1.1 \cdot 10^6$ neurons, previous simulation code shows a reduction in wall-clock time only up to 2048 processes, after which it increases again. Employing the new technology, simulation speed increases up to 8192 processes, enabling the simulation of more than one million neurons with plastic synapses for 1s biological time in 16s wall-clock time. Although the design of the new technology focuses on the regime beyond 10^4 MPI processes, optimizations for small-scale simulations on laptops and workstations make sure that the alterations have no negative impact on the overall performance. While this study considers the concrete use case of the NEST simulator in order to obtain quantitative data, the concepts that are presented here are of general nature: The new data structures and algorithms can be implemented in any neural simulator that abstracts neurons and synapses as individual objects allocated as a whole on a single compute node.

The new simulation kernel also mitigates memory allocation bottlenecks observed in the previous kernel when using non-thread-aware allocators (Ippen et al., 2017). The previous technology stores connections in dynamic containers separated according to source neurons, each container storing very few entries in the case of large networks (Kunkel et al., 2014). Due to this separation many small objects were allocated during network construction. In the extreme case of a full JUQUEEN simulation this means that each of the $\sim 10^8$ synapse objects per compute node plus its container are allocated separately. In the new technology, all synapse objects of the same type on the same thread are stored in a single container irrespective of their source neurons. Connection routines can thus predict the total required memory and allocate it en bloc prior to instantiating the individual connection objects, which leads to a reduced build time, even with default allocators. However, not all connection builders currently available in simulation codes like NEST support this prediction and it needs to be investigated for which connection routines such a strategy is viable. The removed separation according to source neuron also allows us to keep the new data structures simple, relying only on STL vectors, without the need for custom container types. The memory-efficient containers developed for the previous kernel might nevertheless become relevant in particular scenarios not investigated in this chapter.

For large-scale simulations employing thousands of MPI processes, the previous technology does not exhibit good scaling over threads (figure C.4). The main reason for this is the scanning of the large MPI receive buffers, which need to be read completely by all threads. The buffers in the new technology are independent of the number of MPI processes and for a large number of MPI processes hence much smaller than previously. The reduced buffer size not only reduces the absolute time spent in this serial section of the code but also increases the fraction of time spent in fully parallelized sections. Therefore the new kernel better exploits the computational power of JUQUEEN when running 64 threads per compute node than the previous kernel, which shows negligible speed-up from 8 to 16 threads in full JUQUEEN simulations (figure C.4). After speed-up diminishes, 4g also reaches another limit on JUQUEEN: From 32 threads on there is not enough memory available to represent the required data structures. When running 8 threads, 5g reduces the wall-clock time required for the propagation of network dynamics by about 55% for a full JUQUEEN simulation compared to 4g. By increasing the number of threads to 64, which is the maximal number of threads per compute node on JUQUEEN, we achieve an additional reduction by about 60%. In total this reduces the time required

for the propagation of network dynamics by more than 80% from 4g with 8 threads to 5g with 64 threads. At a network size of half a billion neurons, simulating 1 second of biological time requires about 30 minutes of wall-clock time on JUQUEEN using the 4g kernel. The new technology implemented in 5g reduces this time to about 5 minutes on JUQUEEN. Support for massive threading will become even more important with modern CPUs being able to run hundreds of concurrent threads.

The new technology relies on `MPI_Alltoall` to deliver spikes only to the ranks on which they are actually required and it potentially uses multiple communication rounds to communicate all spikes that accumulated in one communication interval. Alternatively one could use `MPI_Alltoallv` to allow ranks to communicate different chunk sizes to different ranks, which would require only a single communication round per communication interval. `MPI_Alltoallv`, however, requires the communication of the respective chunk sizes for all ranks prior to the communication of the actual data, incurring a performance penalty for large-scale simulations (Thakur et al., 2010). Due to the round-robin distribution of neurons across compute nodes and a high number of neurons per compute node, the number of spikes generated per compute node in a single communication interval is fluctuating very little. Using `MPI_Alltoall` with a fixed size of MPI buffers and initiating an additional communication round only when necessary is hence superior to using `MPI_Alltoallv`, which always requires an additional communication step. While `MPI_Alltoall` is feasible at this scale, it relies on global communication, which means that every compute node is needs to take part in every MPI communication. An alternative to this scheme would be the definition of communicators that only include a subset of nodes, thereby possibly reducing communication time. For the round-robin distribution of neurons and the random connectivity in our benchmark model it is unlikely that a randomly picked neuron has a target on a randomly picked process for simulations involving 10^5 compute nodes (cf. section 8.1). But the probability that any of the neurons on a process has at least one target on a randomly picked process is still close to one; so each process needs to communicate to each other process at some point during the simulation. So far it is hence unclear how to exploit communication patterns involving only a subset of nodes, for example neighborhood collectives, to improve performance of the simulator. In particular one needs to investigate how to map the spatial structure of a neuronal network to the topology of modern HPC systems. Since this would most likely imply a non-random distribution of neurons across nodes one needs to replace the current static load balancing via the round-robin distribution of neurons over nodes with alternatives.

Regarding alternative data structures for spike routing, two other options could be considered. The sparse table essentially allows quick access to all connections originating from a single source via a globally valid index, which is the source GID. To reduce memory usage at the expense of computation time, one could replace the sparse table with a vector in which the containers that hold the connections are sorted by source. This vector would only contain entries for the sources which have local targets and would hence not scale with the total number of neurons in the network. Upon receiving a spike one could then use a binary search on this structure to access all relevant connections. Although the complexity of this operation is only $\sim \log N$ for a sorted vector, it nevertheless becomes prohibitively expensive for large network, as this search is required for every spike to be delivered. Instead of a sorted vector, one could use a hash map with $\mathcal{O}(1)$ lookup complexity. This fast access, however, comes at the expense of increased memory usage. The new technology presented here combines the lean memory usage of plain vectors with fast lookup ($\mathcal{O}(1)$) by employing the additional presynaptic data structure.

Future work can further improve the connection infrastructure in several aspects. One of the main drawbacks of not storing connections separated according to their source neurons is the increased size of STDP connection objects. Each of them needs to store the time of the previous spike (8B), required to perform weight updates. Previously, this information only needed to be stored once per source neuron. This overhead leads to an increased memory footprint of connection objects for simulations in which the total number of threads is much smaller than the in-degree of a neuron ($MT \ll K$). One possible remedy is to introduce a new type of spike event that not only includes information to identify the postsynaptic target, but also the time of the previous spike of the presynaptic neuron. This, however, requires changes to the presynaptic data structures responsible for buffering spikes between communication intervals.

The data structures that hold connection information are separated according to threads. In the case that neurons on different threads on the same rank receive connections from the same source, this source needs to send one spike per thread on the target rank; the number of spikes a single neuron emits therefore scales with MT instead of just the number of MPI processes M . This leads to an increase in the amount of data to be communicated and to a growth of the presynaptic data structure with the number of threads per process. To fully exploit the large number of threads supported by modern CPUs, one needs to investigate whether this strict separation of data by threads on a single rank can be relaxed.

During construction of the presynaptic data structures, the strict separation of data by threads is not respected (cf. section 8.3.1), leading to a bottleneck during collocation of connectivity data in the MPI buffers, which contributes significantly to the presim time. The algorithm responsible for collocation can potentially be improved to process these data structures in parallel. Such improvements require detailed profiling of the current implementation and the design of specific thread-parallel memory access patterns for the MPI send buffer.

In total, each process needs to communicate about KN_M individual objects of connectivity data to all other ranks, which totals $\sim 1.5\text{GB}$ for a full JUQUEEN simulation. To reduce the amount of data that need to be communicated one could compress the information prior to the MPI call, which, however, would most likely require the use of `MPI_Alltoallv`, because the sizes of the compressed data need to be communicated in addition to the actual data. Also on the presynaptic side, there is a potential benefit for compressing connection information, as the target lists consume a significant portion of the total available memory for large-scale simulations (cf. figure 8.7; see Morrison et al. (2005) for a simple type of compression). Instead of an STL vector storing the targets of each neuron, one could use a dynamically compressing container that only decompresses target information when needed. Assuming that neurons have average firing rates of 1 Hz, one only needs information about a neuron's targets once in 1000 communication steps for a min delay of 1 ms, or, equivalently, once in 10,000 update steps for a simulation resolution of 0.1 ms. The reduction in memory usage could hence outweigh the costs of having to compress and decompress the corresponding data. To accommodate for the bursty firing of some neuron models, this container should provide efficient caching.

The new connection infrastructure allows for the introduction of models of synaptic plasticity such as STDP with predominantly axonal delays. In previous kernel versions, only fractions of up to 50% axonal delay can be implemented in a straight-forward fashion (see Morrison, Aertsen, and Diesmann, 2007a) as spikes are delivered through the synapses to the postsynaptic targets at the beginning

every communication interval. The contributions of the spikes are then directly incorporated in the ring-buffers of the target neurons. Axonal fractions of the delay of more than 50% could require altering some of the spike contributions retrospectively. With the additional information on the presynaptic side, one can introduce axonal delays, by postponing the time when spikes are added to the MPI send buffer. This extension requires changes to the presynaptic data structures and also entail changes to the user interface, which currently only supports the definition of synaptic transmission delays but not of an axonal fraction; by default the delay is assumed to be purely dendritic.

With the introduction of the new technology, implementations of further biophysical mechanisms such as structural plasticity and the support of continuous interactions between neurons needed to be adapted, as they are tightly coupled to the connection infrastructure. The new kernel of the NEST simulation code encompasses all biophysical mechanisms of the previous kernel.

The new technology supports the creation and deletion of synapses at runtime, which is a prerequisite for models of structural plasticity. However, since connectivity information is now spread over two data structures instead of one, an efficient implementation of particular models of structural plasticity can become more challenging. For the particular case of the structural plasticity described by Diaz-Pier et al. (2016), the new simulation kernel is slightly faster for simulations that employ less than about one hundred MPI processes (section C.4). For larger simulations, the performance degrades faster than previously since the implementation destroys the presynaptic part whenever connections are created or removed. As this structural plasticity model is updated on a time grid much larger than the simulation resolution, typically on the order of seconds, the impact of this naive implementation on the runtime is acceptable. However, the algorithm on which the model is based is not well scalable to thousands of processes, as it collects global information about possible connections on each compute node (Diaz-Pier et al., 2016). The new two-tier connection infrastructure opens the possibility of improved implementations that do not require communication between MPI processes during structural plasticity updates and only operate on the locally available data structures. Further work is required to provide a common framework for scalable structural plasticity models.

Continuous coupling between units, for example via gap junctions, requires the communication of large amounts of data between compute nodes. The large sizes of the MPI buffers lead to significant scaling issues limiting the maximal network size that can routinely be simulated (see, for example, Fig.13 in Hahne et al., 2015). The directed communication of the new technology reduces the size of the MPI buffers. Although a detailed investigation of continuous coupling is outside the scope of the present study, directed communication potentially improves the scalability of simulations employing models with continuous interaction.

As memory consumption is now under control for post-petascale system the focus of future technological developments needs to be on accelerating network construction and simulation for large-scale networks in order to make such simulations more attractive for neuroscientific modeling and robotics applications. Large, distributed simulations currently run at a fraction of real time, with one second of biological time requiring hundreds of seconds of compute time (figure 8.15) prohibiting fast model-development cycles and studies of plasticity and learning. Most simulation time is currently spent on delivering spikes from the MPI receive buffers to the target neurons through the connection infrastructure and synapse objects (figure 8.15), composed of two main contributions. First, plasticity rules in their current implementation require frequent, expensive computations of exponential functions

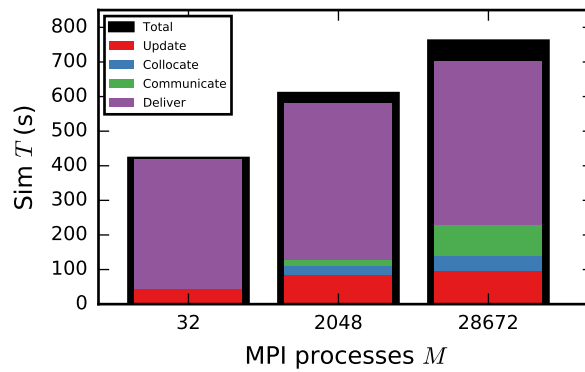


FIGURE 8.15: **Contributions to wall-clock time for a network simulation on different numbers of MPI processes.** Total wall-clock time required for propagation of network dynamics (black), time spend on update of neurons and population of spike register from target lists (red), collocation of MPI send buffer (blue), `MPI_Alltoall` communication (green), reading of MPI buffers and distribution of spikes via synapses to targets (purple) on the JUQUEEN supercomputer. Timings obtained via manual instrumentation of the respective parts of the source code. Same network configuration as in figure 8.7.

as they employ low-pass filtered spike trains. Second, spikes need to be delivered to ring-buffers of the target neurons. The latter operation requires frequent random memory access and consequently suboptimal cache usage. Performance could potentially be improved by removing the spike buffers from the individual neuron objects and using a single buffer for all spikes that need to be delivered to thread-local neurons, thereby improving memory-locality. In addition, all neuron objects are currently allocated by a single thread, which, in systems with non-uniform memory access, can lead to increased memory-access latencies (see also Ippen et al., 2017). To avoid this, neurons and connections need to be created by the thread that will most frequently access the corresponding objects.

For our benchmark networks the time required for communication of spike data is small compared to the time required for spike delivery (figure 8.15). Since the minimal delay in the network is 1.5 ms, communication takes place on a rather coarse time grid. However, when considering other network models, especially models including randomly distributed delays (e.g., Potjans and Diesmann, 2014), the minimal delay can be of the same order of magnitude as the simulation resolution, significantly increasing the relative contribution of communication time to the total wall-clock time. To address these modeling scenarios, alternative communication schemes need to be explored that possibly interleave communication and delivery of spikes by using non-blocking MPI calls.

The network model we consider here represents a worst-case scenario in terms of connectivity, since all pairs of neurons have the same probability of forming a connection. Biological neuronal networks larger than the cortical microcircuit, however, exhibit spatial organization on multiple levels. In a local area for example, the connection probability for a pair of neurons decreases with their distance and long-range connections have a longer transmission delay than short-range connections (Hellwig, 2000; Stepanyants et al., 2009; Markov et al., 2014a). Thus, at the brain scale there is indeed the biophysical basis for the idea indicated above, to map the spatial organization of structured neuronal network models (e.g., Senk et al., 2015; Schmidt et al., 2016) to the topology of the networks of HPC systems. Whether this

would allow, for example, the use of MPI neighborhood collectives needs to be investigated. Kozloski and Wagner (2011) take the idea to the extreme and abandon the concept of a dynamics on a graph. They decompose neural tissue into regular blocks in three-dimensional space and allow communication only between adjacent surfaces of blocks. So far however it remains unclear whether a similar strategy can be used for more efficient simulations of large-scale networks of simple neuron models.

As the scale of HPC systems increases towards the exascale, so does the probability of device failure (Thakur et al., 2010; Dongarra et al., 2011; Cappello et al., 2014). To avoid wasting precious computational resources upon device failure, fault-tolerant communication options need to be explored. In particular support for malleability, the possibility of changing the number of MPI processes or threads per process during a running simulation, would improve the fault tolerance of the application upon failure of a single core and should be considered in future developments. Since for large-scale simulations, one typically uses the whole available memory per compute node, it is most likely not possible to recover from failure of a complete compute node, even with a malleable program. To cope with this type of failure, one needs to explore options for regularly storing the progress of the simulation, called checkpointing, in order to be able to restart the simulation from a given point after the crash.

While in our particular network model the propagation of single-cell dynamics only accounts for a small fraction of the total wall-clock time required for the simulation (figure 8.15), this ratio can change if one considers more complex neuron models, for example models with non-linear sub-threshold dynamics or multiple compartments. To address this issue, the possibility of using accelerators available in modern HPC systems, for example general-purpose graphics processing units or field-programmable gate arrays, needs to be investigated. Here, efficient memory access of the accelerator to main memory is critical.

The simulation technology presented in this chapter exploits the available main memory of contemporary supercomputers and shows perfect scalability in a weak-scaling scenario. Now the focus lies on accelerating simulations in the presence of various forms of network plasticity.

Established simulation software, like discussed in this chapter, allows researchers in computational neuroscience to efficiently simulate large-scale network models. In the domain of functional networks, simulating the network dynamics is, however, only one part: meaningful input to the network model is equally important. So far, most studies in this area are conducted with custom simulation scripts and manually implemented tasks, making it hard for other researchers to reproduce and build upon previous work and nearly impossible to compare the performance of different learning architectures. In the next chapter, we present a novel approach to solve this problem, connecting benchmark tools from the field of machine learning and state-of-the-art neural network simulators from computational neuroscience.

Chapter 9

Interfacing neural network simulators and the OpenAI Gym

This chapter is based on the following publication:

Jakob Jordan, Philipp Weidel, and Abigail Morrison (2017). “Closing the loop between neural network simulators and the OpenAI Gym”. In: *ArXiv e-prints*. arXiv: 1709.05650 [q-bio.NC]

Equal contribution by Jakob Jordan & Philipp Weidel

Author contributions:

Philipp Weidel and the author performed the work concertedly with equal shares on all parts of the study. Abigail Morrison supervised the work and all authors contributed to the writing of the manuscript.

9.1 Introduction

The last decade has witnessed major progress in the field of machine learning, moving from small-scale toy problems to large-scale real-world applications including image (Krizhevsky, Sutskever, and Hinton, 2012) and speech recognition (Hinton et al., 2012a), complex motor-control tasks (Mnih et al., 2016) and playing (video) games at super-human performance (Mnih et al., 2015; Silver et al., 2016). This progress has been driven mainly by an increase in computing power, especially by training deep networks on graphics processing units (Raina, Madhavan, and Ng, 2009), and conceptual breakthroughs like layer-wise pretraining (Hinton and Salakhutdinov, 2006a; Bengio et al., 2007) or dropout (Hinton et al., 2012b). Even so, this rate of progress would not have been possible without the wide availability of high-performance ready-to-use tools, e.g., Torch (Collobert, Bengio, and Mariéthoz, 2002), Theano (James et al., 2010), Caffe (Jia et al., 2014), TensorFlow (Abadi et al., 2016) and standardized benchmarks and learning environments, such as the MNIST (LeCun, Cortes, and Burges, 1998), CIFAR (Krizhevsky and Hinton, 2009) and ImageNET (Deng et al., 2009) datasets, and the MuJoCo (Todorov, Erez, and Tassa, 2012), ALE (Bellemare et al., 2015) and OpenAI Gym (Brockman et al., 2016) toolkits. While ready-to-use tools allow researchers to focus on important aspects rather than basic implementation details, standardized benchmarks can guide the community as a whole towards promising approaches, as for example in the case of convolutional networks through the ImageNET competition (Russakovsky et al., 2015).

Similarly, researchers in the field of computational neuroscience have benefited from the increase of computational power and achieved many conceptual breakthroughs over the last decade, with a plethora of new neuron, synapse and network models being developed. Thanks to a variety of software projects, researchers have access to simulators for all scales of neural systems from molecular simulations (Wils and De Schutter, 2009) over complex neuron (Carnevale and Hines, 2006; Bower and Beeman, 2007) and network models (Gewaltig and Diesmann, 2007; Goodman and Brette, 2009; Bekolay et al., 2013) to whole brain simulations using neural fields (Sanz Leon et al., 2013). However, in computational neuroscience no generally accepted set of benchmarks exist so far (but see Gerstner and Naud (2009)). While it is desirable to compare different neural network models with respect to biological plausibility, explanatory power and functional performance, only the latter lends itself to the definition of quantitative benchmarks.

One particular area in which the lack of standardized benchmarks is apparent is research into reinforcement learning (RL) in neurobiological substrates. Inspired by behavioural experiments, RL is concerned with the ability of organisms to learn from previous experiences to optimize their behavior in order to maximize reward and avoid punishment (see, e.g., Sutton and Barto (1998)). RL has a long tradition in the field of machine learning which has led to several powerful algorithms, such as SARSA and Q-learning (Watkins, 1989). Similarly, a large variety of neurobiological models have been proposed in recent years (Vasilaki et al., 2009; Izhikevich, 2007; Urbanczik and Senn, 2009; Potjans, Morrison, and Diesmann, 2009; Frémaux, Sprekeler, and Gerstner, 2010; Jitsev, Morrison, and Tittgemeyer, 2012; Frémaux, Sprekeler, and Gerstner, 2013; Rasmussen and Eliasmith, 2014; Friedrich, Urbanczik, and Senn, 2014; Rombouts, Bohte, and Roelfsema, 2015; Baladron and Hamker, 2015; Aswolinskiy and Pipa, 2015; Friedrich and Lengyel, 2016; Rueckert et al., 2016). However, only a small proportion of these rely on publicly available simulators and all of them employ custom built environments. Even for fairly simple environments, this has led to a situation where different network models are difficult to compare

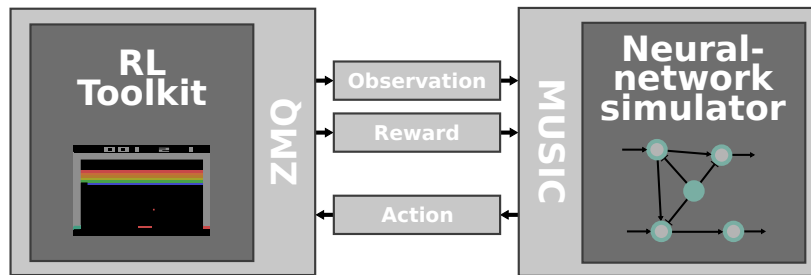


FIGURE 9.1: **Interfacing RL toolkits with neural network simulators.** The RL toolkit (left) is responsible for emulating an environment that provides observations and rewards which are communicated via ZeroMQ sockets and MUSIC adapters (middle) to a neural network simulator (right). Concurrently, the activity of the simulated neural network is transformed to an action and fed back to the RL toolkit.

and reproduce, thus creating a fragmentation of research efforts. Instead of building upon and extending existing models, researchers are forced to spend too much time on recreating basic methods for custom implementations.

This issue has led to the Human Brain Project (HBP) (Human Brain Project, 2014) dedicating significant resources of a subproject (SP10, Neurorobotics) to the development of the necessary infrastructure that allows users to conduct robotic experiments in virtual environments and connect these to their neural network implementations with an easy to use web interface. This approach however, specifically addresses the need of researchers developing neuronal controllers for robotic applications.

In contrast, the OpenAI Gym (Brockman et al., 2016) provides a rich and generic collection of standardized RL environments developed to support the machine learning community in evaluating and comparing algorithms. All environments are accessible via a simple, unified interface, that requires an agent to supply an action and returns an observation and reward for its current state. The toolkit includes a range of different environments with varying levels of complexity ranging from low-dimensional fully discrete (e.g., *FrozenLake*) to high-dimensional fully continuous tasks (e.g., *Humanoid*). Consistency of the OpenAI Gym environments across different releases supports researchers in reproduction and extension of previous work and allows systematic benchmarking and comparison of learning algorithms and their implementations. The easy accessibility of different tasks fosters progress by allowing researchers to focus on learning algorithms instead of basic implementation details of particular environments and provokes researchers to evaluate the performance of their algorithms on many different tasks.

To make this comprehensive resource available to the computational neuroscience community, we developed a toolchain to interface neural network simulators with the OpenAI Gym. Using this toolchain, researchers can rely on well-tested, high-performance simulation engines to power their models and evaluate them against a curated set of standardized environments, allowing more time to focus on neurobiological questions.

In the next section we introduce additional pre-existing components on which our toolchain relies, and afterwards discuss how it links the different tools. We demonstrate its functionality by implementing a neural actor-critic in NEST (Gewaltig and Diesmann, 2007) and successfully training it on two different environments from the OpenAI Gym.

9.2 Pre-existing components

NEST is a neural simulator designed for the efficient simulation of large-scale networks of simple neuron models with biophysically realistic connectivity. The simulation kernel scales from small simulations on a laptops to super computers, with the largest simulation containing about 10^9 neurons and 10^{13} synapses, corresponding to about 10% of the human cortex at the resolution of individual cells and connections (Kunkel et al., 2014). NEST is actively developed and maintained by the NEST initiative in collaboration with the community and freely available under the GPLv2 and is supported by the HBP with the explicit aim of widespread long-term availability and maintainability.

While the network implementation that we present in the results section relies on the NEST simulator, the toolchain can also be used with other simulators that support the MUSIC library, for example NEURON (Carnevale and Hines, 2006). The Multi-Simulation Coordinator is a multi-purpose middleware for neural network simulators built on top of MPI (Message Passing Interface) that enables online interaction of different simulation engines (Djurfeldt et al., 2010b). MUSIC provides named MPI channels, referred to as MUSIC ports, which allow the user to set up communication streams between several processes. While originally intended to distribute a single neural network model across different simulators, the MUSIC library can also be used to connect neural simulators with robotic applications.

For this purpose the ROS-MUSIC Toolchain (RMT) (Weidel et al., 2016) was recently developed, providing an interface from MUSIC to the Robotic Operating System (ROS) (Quigley et al., 2009). ROS is the most popular middleware in the robotic community which can interact with many robotic simulators and hardware platforms. The RMT allows exchange of well-defined messages between ROS and MUSIC via stand-alone executables, so called adapters, that were designed with a focus on modularity. The toolchain contains several different adapters each performing a rather simple operation on streams of inputs (e.g., filtering). By concatenating several adapters, the overall transformation of the original data can become more complex, for example converting high-dimensional continuous data (e.g., sensory data) to low-dimensional discrete data (e.g., action potentials) or vice-versa. More information and introductory examples can be found on GitHub.¹

9.3 Results

To enable the online interaction of neural network simulators and the OpenAI Gym we rely on two different libraries, MUSIC, to interface with the neural simulator, and ZeroMQ (Hintjens, 2013) to exchange messages with the environment simulated in the OpenAI Gym. In the following we describe these two parts of the toolchain and demonstrate their functionality by interfacing a neural network simulation in NEST with two different environments.

9.3.1 Extending the ROS - MUSIC toolchain

We extended the RMT by adding adapters that support communication via ZeroMQ following a publish-subscribe pattern. ZeroMQ is a messaging library that allows applications to exchange messages at runtime via sockets. Continuously developed by a large community, it offers bindings for a variety of languages including C++ and

¹<https://github.com/incf-music/ros-music-adapters>

Python, and supports most operating systems. A single communication adapter of the RMT sends (receives) data via a ZeroMQ socket and receives (sends) data via a MUSIC port. While the adapters can handle arbitrary data, we defined a set of specialized messages in JSON format (see supplementary material) specifically designed to communicate observations, rewards and actions as discrete or continuous real-valued variables of arbitrary dimensions, as used in the OpenAI Gym. We chose the JSON format due to its simplicity, easy serialization and broad platform support.

In addition to the ZeroMQ adapters dedicated for communication with MUSIC, we developed several further adapters that can perform specific transformations of the data. As discussed above, environments can be defined in continuous or discrete spaces with arbitrary dimensionality. To generate the required closed-loop functionality, the observations provided by the environment must be consistently transformed to a format that can be fed into neural network simulations. Conversely, the activity of the neural network must be interpreted and transformed into valid actions which can be executed in the environment.

A standard way to address the first issue is to introduce so called *place cells*. Each of these cells is tuned to a preferred (multidimensional) observation, i.e., is highly active for a specific input and less active for other inputs (Frémaux, Sprekeler, and Gerstner, 2013). The dependence of the activity of a single place cell on observations is described by its tuning curve, often chosen as a multidimensional Gaussian. To perform the transformation of observations to activity of place cells, we implemented a *discretize adapter* that allows users to specify the position and width of the tuning curves of an arbitrary number of place cells. While having a certain biological plausibility (Moser, Kropff, and Moser, 2008), one disadvantage of this approach is that the number of place cells required to cover the whole observation space evenly scales exponentially in the number of dimensions of the observation. For observations with a small number of dimensions, however, this approach is very suitable.

To perform action selection, we added several adapters that can, respectively, select the most active neuron (*argmax adapter*), threshold the activity across neurons to create a binary vector (*threshold adapter*) or linearly combine the activity of neurons across many input channels (*linear decoder*). Depending on the type of action required (discrete/continuous) by the environment, the user can select a single one or a combination of these. See the documentation of the RMT for detailed specifications of the adapters.

In general, we followed the design principle behind the RMT and developed modular adapters. This makes each individual adapter easy to understand and enables users to quickly extend the toolchain with their own adapters. By combining several adapters, the RMT allows arbitrarily complex transformations of the data and can hence be applied to many use-cases.

9.3.2 ZeroMQ wrapper for the OpenAI Gym

The second part of the toolchain is a Python wrapper around the OpenAI Gym that exposes ZeroMQ sockets for communicating actions, observations and rewards (see figure 9.1). An environment in the OpenAI Gym is updated in steps. In each step, an agent needs to provide an action and receives an observation and reward for its current state. The wrapper consists of four different threads that coordinate: (i) performing steps in an environment, (ii) receiving actions via a ZeroMQ SUB socket, (iii) publishing observations via a ZeroMQ PUB socket and (iv) publishing rewards via a ZeroMQ PUB socket. Before spawning the threads, the wrapper starts a user-specified environment and creates the necessary communication buffers. The

thread coordinating the environment reads actions from the corresponding buffer, performs single steps in the environment and updates the observation and reward buffers based on the return values of the environment. Upon detecting that a single episode has ended, e.g., by an agent reaching a certain goal position, it resets the environment and allows a user-specified break before starting another episode. The communication threads continuously send(receive) messages via ZeroMQ and read from(write to) the corresponding buffers. All threads can be run with different update intervals, for example, to slow down movement of the agent by performing steps on a coarse time grid whilst continuously receiving action choices from the neural network simulation running on a fine time grid. The user can specify a variety of parameters via a configuration file in JSON format (see supplementary material). See the documentation for detailed specifications of the wrapper.

9.3.3 Applications

To demonstrate the functionality of the toolchain, we implemented a neural network model with actor-critic architecture in NEST and trained it on two different environments simulated in the OpenAI Gym. In the first task, the agent needs to learn to perform a sequence of actions in order to reach the top of a hill in a continuous environment. The second task is a classical grid-world in which an agent needs to learn to navigate to a goal position in a two-dimensional discrete environment with obstacles. We first describe the neural network architecture and learning rule and afterwards discuss the network's performance on the two tasks.

Neural network implementation

We consider a temporal-difference learning algorithm (Sutton and Barto, 1998) implemented as an actor-critic architecture, originally using populations of spiking neurons (Frémaux, Sprekeler, and Gerstner, 2013). We translated the spike-based implementation to rate neurons, mainly to simplify the implementation by avoiding issues arising from noise introduced by spiking neuron models (Potjans, Diesmann, and Morrison, 2011; Frémaux, Sprekeler, and Gerstner, 2013). The neuron dynamics we considered here are determined by the following stochastic differential equation:

$$\tau \frac{dz_i(t)}{dt} = -z_i(t) + \mu_i + f(h_i(t) - \theta_i) + \xi_i(t), \quad (9.1)$$

where τ is some positive time constant, μ_i a baseline activity level, $f(\cdot)$ some (arbitrary) activation function, $h_i(t)$ a time dependent input field, θ_i an input threshold and $\xi_i(t)$ white noise with a certain standard deviation σ_ξ . The input field $h_i(t)$ is determined by the activity of other neurons according to $h_i(t) = \sum_j w_{ij} z_j(t)$, with w_{ij} denoting the strength of the connection (weight) from neuron j to neuron i . Here we will exclusively consider activation functions of the form $f(x) = x$ (linear case), and $f(x) = \Theta(x)x$ (threshold-linear case, "relu"). These models have recently been added to the NEST simulator. Their dynamics are solved on a fixed time-grid by a stochastic-exponential-Euler method with a step size determined by the resolution of the simulation. For more details on the model implementation see (Hahne et al., 2016).

The input layer is a population of threshold-linear rate neurons which receive inputs through MUSIC and encode observations from the environment (see figure 9.2). Via plastic connections these place cells project to a single neuron representing the

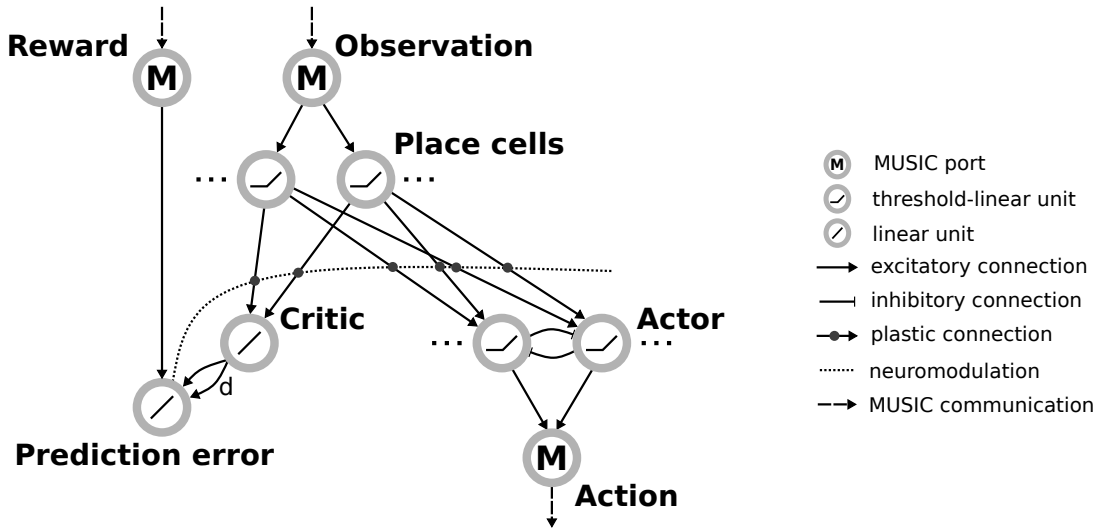


FIGURE 9.2: **Actor-critic architecture for RL with rate neurons.** Observations are communicated via a MUSIC input port to a population of place cells. These project on the one hand to a critic unit and on the other hand to actor units arranged in a winner-take-all circuit. The critic and an additional MUSIC input port project to a unit representing the reward-prediction error that modulates the plasticity between place cells and critic and actors, respectively. The actor units project to a MUSIC output port encoding the selected action.

value that the network assigns to the current state (the “critic”). An additional neuron calculates the reward-prediction error by combining the reward received from the environment with input from the critic. Plasticity of the projections from inputs to the critic is modulated by this reward-prediction error (see below).

In addition, neurons in the input layer project to a population of neurons representing the available actions (the “actor”). To enforce selection of a specific action, the actor units are arranged in a winner-take-all (WTA) circuit. This is implemented by recurrent connections between actor units that correspond to short-range excitation and long-range inhibition, depending on the similarity of the action that actor units encode. The activity of actor units is transformed to a valid action and communicated to the environment via the RMT.

To derive a learning rule for the critic, we followed similar steps as in (Frémaux, Sprekeler, and Gerstner, 2013) applied to rate models (equation (9.1)). The critic activity should approximate a continuous-time value function defined by (Doya, 2000)

$$V^\pi(s(t)) := \int_{t'}^{\infty} r(s^\pi(t')) e^{-\frac{t'-t}{\tau_r}} dt'. \quad (9.2)$$

Here $s(t)$ denotes the state of the agent at time t , $r(s^\pi(t))$ denotes the reward obtained in state $s(t)$, τ_r a discounting factor for future rewards and π the agent’s policy. To achieve this, we define the following objective function which should be minimized by gradient descent on the weights from inputs to the critic:

$$E(t) = \frac{1}{2}(V^\pi(t) - z(t))^2, \quad (9.3)$$

where $z(t)$ represents the activity of the critic unit. By performing gradient descent on equation (9.3), using a self-consistency equation for $V^\pi(t)$ from the derivative of

equation (9.2) and bootstrapping on the current prediction for the value (see supplementary material, Doya (2000) and Frémaux, Sprekeler, and Gerstner (2013)), we obtain the following local Hebbian three-factor learning rule that approximately minimizes the objective function (equation (9.3)):

$$\Delta w_j = \eta \delta(t) x_j(t) \Theta(z(t) - \theta_{\text{post}}) , \quad (9.4)$$

where η is a learning rate, $x_j(t)$ represents the activity of the j th place cell, $\Theta(\cdot)$ the Heaviside function and θ_{post} a parameter that accounts for noise on the postsynaptic unit (see supplementary material for details). The term $\delta(t) = \dot{v}(t) + r(t) - \frac{1}{\tau_r}v(t)$ corresponds to the activity of the reward-prediction error unit, acting as a neuromodulatory signal for the Hebbian plasticity between the presynaptic (x_j) and postsynaptic (z) units. To avoid explicit calculation of the derivative, we rewrite $\delta(t)$ as:

$$\delta(t) \approx \left(\frac{1}{d} - \frac{1}{\tau_r} \right) v(t) - \frac{1}{d} v(t-d) + r(t) . \quad (9.5)$$

To approximate the derivative we hence implement two connections from the critic to the reward-prediction error unit: one instantaneous, and one with delay $d > 0$.

To learn an optimal policy, we exploit that the actor units follow the same dynamics as the critic. Similar to Frémaux, Sprekeler, and Gerstner (2013), we hence apply the same learning rule to the connections between the inputs and the actor units. In order to assure that at least one actor unit is active, thus preventing a deadlock, we introduce a minimal weight for each connection between input and output units and add input noise to the actor units.

Mountain Car

As an example of an environment with continuous states, we consider the *Mountain-Car* environment. The task is to steer a toy vehicle that starts at a valley between two hills to the top of the right one (figure 9.3A, inset). To make the task more challenging, the car's engine is not strong enough to reach the top in one go, so the agent needs to learn to gain momentum by swinging back and forth between the two hills. A single episode in this environment starts when the agent is placed in the valley and ends when it reaches the final position on the top of the right hill. The state of the agent is described by two continuous variables: the x-position $x(t)$ and the x-velocity $\dot{x}(t)$. The agent can choose from three different discrete actions that affect the velocity of the vehicle (accelerate left, no acceleration, accelerate right). It receives punishment from the environment in every step; the goal is to minimize the total punishment collected over the whole episode. Since this is difficult to implement in an actor-critic architecture (Potjans, Diesmann, and Morrison, 2011), we provide additional reward when the agent reaches the final position.

To translate the agent's current state into neuronal activity, we distribute 25 place cells evenly across the two dimensional plane of possible positions and velocities using the *discretize adapter* of the RMT. The actor is implemented by a WTA circuit of three units as described in section 9.3.3. The activity of these units is transformed into an action via the *argmax adapter* (section 9.3.1).

Initially, the agent explores the environment by selecting random actions. Due to the WTA circuit dynamics, a single action stays active over an extended period of time. The constant punishment gradually decreases the weights from the place cells to the corresponding actor unit, eventually leading to another actor unit becoming

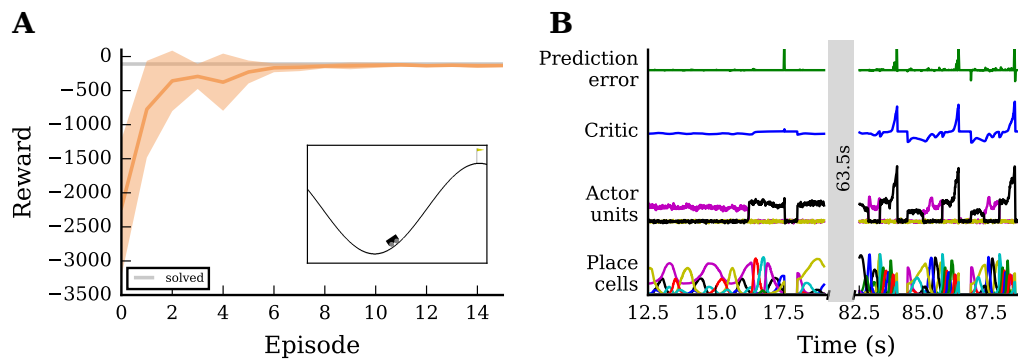


FIGURE 9.3: **Network performance on an environment with continuous states and discrete actions.** **A:** Reward obtained by the agent per episode averaged over 10 simulations with different seeds (solid orange line). Orange band indicates $\pm$ one standard deviation. Light gray line marks average reward per episode for which the environment is considered solved. Inset: screenshot of the environment with agent (stylized vehicle), environment with valley and two hills (black solid line) and goal position (yellow flag). The agent is close to the starting position at the trough. **B:** Activity traces of place cells (bottom), actor units (second from bottom), critic unit (second from top) and reward-prediction-error unit (top). Shown are neural activities during 6.5 s early (left) and late (right) in the simulation.

active (figure 9.3B, left). After a while, the agent reaches the goal by performing actions that have not been significantly punished. For this task the stable nature of the WTA is advantageous, causing the agent to perform the same action repeatedly allowing efficient exploration of the state space. After the agent has found the goal once, the number of steps spent on exploring actions in the following episodes is much smaller. From the sixth episode on, the performance of the agent is already close to optimal (figure 9.3A). After learning for about 10 episodes, the agent's performance has converged. The value of the final state is successfully propagated backwards over different states, leading to a ramping of activity of the critic unit from the start of an episode to the end (figure 9.3B, right).

Since the OpenAI Gym offers a variety of environments, we trained the same network model on an additional task with different requirements.

Frozen Lake

As a second application, we chose the *FrozenLake* environment consisting of a discrete set of 16 states arranged in a four-by-four grid (figure 9.4A, inset). Each state is either a start state (S), a goal state (G), a hole (H) or a frozen state (F). From the start position, the agent has to reach the rewarded state by navigating over the frozen states without falling into holes which would reset the agent to the starting position. In each step, the agent can choose from four different actions: move west, move north, move east and move south. Usually, the tiles are "slippery", i.e., there is a chance that a random action is executed irrespective of the action chosen by the agent. However, to simplify learning for demonstration purposes, we turned this feature off. Upon reaching the goal, the agent receives a reward of magnitude one. Since the optimal path involves six steps from start to goal, the theoretical optimal reward per step is ~ 0.16 . To encourage exploration, the agent receives a small punishment in each state, and additionally, to speed up learning, the agent is punished for falling into holes.

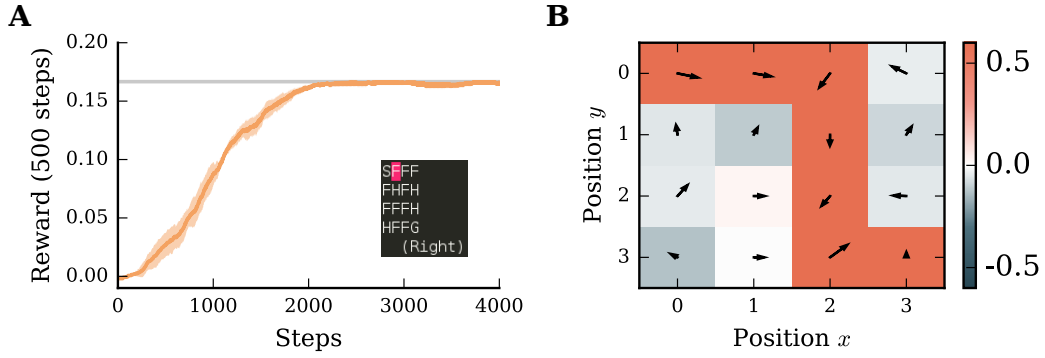


FIGURE 9.4: **Network performance on a grid-world environment.** **A:** Average reward collected by the agent over the next 500 steps (orange solid line) averaged over 5 simulations. Orange band indicates $\pm$ one standard deviation. Gray line: theoretical optimum. Inset: screenshot of the environment with start state (S), frozen states (F), holes (H) and goal state (G). The position of the agent is indicated in pink. **B:** The learned policy and value map of the environment. Red colors indicate positive, blue colors negative values. Arrows indicate the preferred direction of movement.

Unlike in the continuous *MountainCar* environment, the tuning curves of place cells do not overlap in the discrete case, leading to sharp transitions in the network activity. This leads to severe issues for associating values and actions with the respective states. To address this problem we introduced a simple eligibility trace by evaluating the activity of the pre- and post synaptic units in the learning rule with a small delay δt (see supplementary material). With this addition, the network model is able to find the optimal solution for this task within roughly 2000 steps (figure 9.4A). It also learns to associate holes with punishment and frozen states with reward if they are on the path to the goal (figure 9.4B). Although there are two possible paths to the goal, the agent prefers the path with less corners, since it is easier to navigate using a WTA circuit.

9.4 Derivation of the learning rule

We consider continuous time, continuous states and continuous actions and follow similar steps as Doya (2000) and Frémaux, Sprekeler, and Gerstner (2013). Starting from the continuous-time value function

$$V^\pi(\mathbf{s}(t)) := \int_t^\infty r(\mathbf{s}^\pi(t')) e^{-\frac{t'-t}{\tau_r}} dt', \quad (9.6)$$

we take the derivative with respect to t to arrive at a self-consistency equation for V :

$$\dot{V}^\pi(t) := \frac{dV^\pi(t)}{dt} = -r(t) + \frac{1}{\tau_r} V^\pi(t). \quad (9.7)$$

To implement temporal-difference learning in a neural network architecture, we would like to approximately represent the true value $V^\pi(t)$ of the state at time t by the rate $z_i(t)$ of a critic neuron. This activity will depend on the activity of input units, i.e., place cells, and the weights between inputs and critic. With initially random weights the self-consistency criterion will not be fulfilled, and will have a finite

error $\delta(t)$:

$$\delta(t) = \dot{z}_i(t) + r(t) - \frac{1}{\tau_r} z_i(t). \quad (9.8)$$

We now define an objective function that should be minimized by gradient descent on the weights:

$$E(t) = \frac{1}{2} (V^\pi(t) - z_i(t))^2. \quad (9.9)$$

We take derivative with respect to w_{ij} and use the self-consistency equation (9.7):

$$\begin{aligned} \frac{\partial E(t)}{\partial w_{ij}} &= -(V^\pi(t) - z_i(t)) \frac{\partial z_i(t)}{\partial w_{ij}} \\ &= -(\tau_r \dot{V}^\pi(t) + \tau_r r(t) - z_i(t)) \frac{\partial z_i(t)}{\partial w_{ij}} \\ &\approx -\underbrace{(\tau_r \dot{z}_i(t) + \tau_r r(t) - z_i(t))}_{\tau_r \delta(t)} \frac{\partial z_i(t)}{\partial w_{ij}} \end{aligned} \quad (9.10)$$

Here we have replaced $\dot{V}^\pi(t)$ with $\dot{z}_i(t)$ in the last line. For a discussion of the validity of this approximation and the convergence of the learning rule, see Frémaux, Sprekeler, and Gerstner (2013). To perform gradient descent on the objective function, we hence need to change the weights according to

$$\begin{aligned} \Delta w_{ij} &:= -\eta' \frac{\partial E(t)}{\partial w_{ij}} \\ &= \eta \delta(t) \frac{\partial z_i(t)}{\partial w_{ij}}, \end{aligned} \quad (9.11)$$

where we introduced $\eta = \eta' \tau_r$. We hence need to determine the derivative of the critic activity with respect to the weights between inputs and critic $\frac{\partial z_i(t)}{\partial w_{ij}}$.

We start from the differential equation describing the dynamics of a threshold-linear rate neuron, and assume the noise to be small, i.e., we drop the term $\xi_i(t)$. Without loss of generality, we assume $\mu_i = 0, \theta_i = 0$. The dynamics are then given by the solution to

$$\tau \frac{dz_i(t)}{dt} = -z_i(t) + g \Theta \left(\sum_j w_{ij} x_j(t) \right) \left(\sum_j w_{ij} x_j(t) \right). \quad (9.12)$$

Variation of constants yields the general solution as a convolution equation:

$$z_i(t) = g \left(\left(\Theta(\dots) \left(\sum_j w_{ij} x_j(\cdot) \right) \right) * \kappa(\cdot) \right) (t), \quad (9.13)$$

where we have introduced the filter kernel $\kappa(t) = \frac{1}{\tau} e^{-\frac{t}{\tau}} \Theta(t)$. We now take the derivative with respect to w_{ij} . While the derivative at $\sum_j w_{ij} x_j(t) = 0$ is technically

not defined, we follow the usual convention and set it to zero. This yields

$$\frac{\partial z_i(t)}{\partial w_{ij}} = \begin{cases} g(x_j * \kappa)(t) & \text{if } \sum_j w_{ij} x_j(t) > 0 \\ 0 & \text{else} \end{cases} \quad (9.14)$$

By combining equation (9.11) with equation (9.14), we obtain the following learning rule:

$$\Delta w_{ij} = \begin{cases} \eta \delta(t) g(x_j * \kappa)(t) & \text{if } \sum_j w_{ij} x_j(t) > 0 \\ 0 & \text{else} \end{cases} \quad (9.15)$$

By choosing the time constant of critic and actors small, we effectively remove the filtering of the presynaptic activity ($\lim_{\tau \rightarrow 0} \kappa(t) = \delta(t)$) and hence ignore it. To simplify this equation further, we rewrite it as a condition on the rate of the postsynaptic neuron by observing that $z_i(t) > 0$ iff $\sum_j w_{ij} x_j(t) > 0$. To implement exploration for similar inputs to all output units we add noise to the activity of the actor units. We only consider a postsynaptic neuron active, if its activity is larger than some threshold θ_{post} . This leads to the following form for the learning rule:

$$\Delta w_{ij} = \eta \delta(t) g x_j(t) \Theta(z_i(t) - \theta_{\text{post}}), \quad (9.16)$$

where we $\Theta(\cdot)$ denotes the Heaviside step function defined as:

$$\Theta(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{else} \end{cases}$$

To implement a simple type of eligibility trace, we introduce an additional parameter δt that can delay the activity of the pre- and post-synaptic units in the learning rule:

$$\Delta w_{ij} = \eta \delta(t) g x_j(t - \delta t) \Theta(z_i(t - \delta t) - \theta_{\text{post}}), \quad (9.17)$$

9.5 Conclusion

In this chapter, we have presented a toolchain that closes the loop between the OpenAI Gym and neural network simulators. We demonstrated the functionality of the toolchain by implementing an actor-critic architecture in NEST and evaluating its performance on two different environments. The performance of the network quickly reached near-optimal performance on these two tasks.

Combining neural network simulators with RL toolkits responds to the growing need of researchers to provide neural network models with rich, dynamic input. Compared to creating customized environments to this end, using readily available tools is easier, often computationally more efficient, and most importantly, supports reproducible science. In addition, having the OpenAI Gym environments as common benchmarks in both fields encourages comparison between traditional machine learning and biologically plausible implementations. In contrast to models presented in previous studies, our toolchain makes it easy for other researchers to extend our implementation of an actor-critic architecture to other environments, replace neurons models or explore alternative learning rules.

While the toolchain currently only supports the OpenAI Gym, the extension to other toolkits is simple due to a modular design of the wrapper. The RMT can

be found on GitHub and is available under the GPLv3. The OpenAI Gym ZeroMQ wrapper is also available via GitHub under the MIT license. A complementary development to the work presented here is provided by SPORE, a framework for reward-based learning with spiking neurons in the NEST simulator.² It provides support for synapse models with time-driven updates, additional support for recording and evaluating traces of neuronal state variables and introduces MUSIC ports for communicating rewards to a running simulation.

With the work presented here we enable researchers to build more easily upon previous studies and evaluate novel models. We hope this boosts the progress in computational neuroscience in uncovering the biophysical mechanisms involved in learning complex tasks from delayed rewards.

²<https://github.com/IGITUGraz/spore-nest-module>

Part III

Conclusion & Outlook

Chapter 10

Conclusion

In the following we summarize our key findings regarding the specific questions presented in chapter 4.

Biophysical sources of stochasticity for approximate inference in neural circuits

In chapter 5 we investigated whether **deterministic networks with a finite number of external noise sources can approximate sampling in Boltzmann machines**. Our main findings can be summarized as follows:

- Networks of deterministic units with private Gaussian additive noise can approximate sampling from Boltzmann distributions, but exhibit a finite deviation from the target distribution even for long sampling duration. This is caused by the single-unit dynamics deviating from those assumed in Boltzmann machines.
- For a limited number of independent noise sources, the noise across sampling units is correlated due to shared sources. These shared-input correlations impair the ability of sampling networks to accurately represent target distributions.
- For noise generated by recurrent networks, shared-noise correlations are suppressed by activity correlations, leading to a sampling performance comparable to the case of private Gaussian noise, even if the number of sources is severely limited.
- The impact of shared-noise correlations depends on the entropy of the target distribution: While for low entropies, shared-input correlations have little influence, for high entropy, they significantly bias the sampling process.
- Small noise networks can supply much larger functional networks with effectively independent noise.
- Our approach is directly transferable to practical applications of generative models and sampling-network implementations based on spiking neuron models.

These results demonstrate that finite, recurrent neural networks can effectively serve as a variable size pool of stochasticity for sample-based approximate computations in neural substrates in vivo and in silico.

The suppression of shared-input correlations arises from a specific correlation structure in recurrent networks with inhibitory feedback (Renart et al., 2010a; Tetzlaff et al., 2012a). In order to exploit the approach discussed above in neuromorphic implementations of approximate inference, we hence investigated in chapter 6

in how far **network heterogeneities in neuromorphic hardware substrates influence correlations in recurrent networks**. Through emulations of spiking networks on the neuromorphic system Spikey (Pfeil et al., 2013b), we found that:

- Even in highly heterogeneous systems with fixed-pattern and temporal noise, decorrelation by inhibitory feedback dynamically suppresses correlations arising from shared inputs.
- However, contrary to intuition, the effectiveness of this suppression depends on the level of heterogeneity in recurrent networks.
- Software simulations with full control over the specific forms of heterogeneities, support the results obtained from the hardware substrate.
- A linear model that mimics network heterogeneities as changes in the effective coupling strength between neurons captures the main effects of our findings.

These results lie the foundation for employing recurrent networks as efficient noise sources for neuromorphic implementations of approximate inference. Neuromorphic mixed-signal systems emulate network dynamics orders of magnitude faster than realtime (Schemmel et al., 2010b), increasing the need for efficient noise generation, but offering unprecedented opportunities for efficient physical inference machines.

With approximate inference in Boltzmann machines serving as a popular model for brain computation, our findings so far lend further biophysical plausibility to circuit-level models and support neuromorphic implementations. However, efficient probabilistic computations, in particular inference via constraints, relies on the particular form of the Boltzmann distribution. We hence investigated in chapter 7 whether **fast inference via constraining network dynamics is possible in non-Boltzmann networks**. As an analytic expression for the stationary distribution is only available for Boltzmann machines, we introduced an algorithm to construct the transition matrix of a Markov chain. This allowed us to compute the stationary distribution of small networks with arbitrary connectivity and activation functions. Our results show that:

- Replacing the logistic activation function with a shifted error function arising from Gaussian additive noise leads to a loss of detailed balance, even for symmetric networks. Consequently, these network exhibit a nonzero net flow of probability mass between different states even under stationary conditions.
- Constraining network dynamics in networks of deterministic units with Gaussian noise does not allow sampling of the exact conditional distributions, although a close approximation is possible in certain parameter regimes.
- Synaptic stochasticity, in our model variability in synaptic weights, leads to state-dependent noise, and to stationary distributions that are significantly different from Boltzmann distributions even for matched parameters.
- In networks with stochastic synapses, constraining network dynamics does not allow sampling from the exact conditionals.
- Networks with stochastic synapses and bipolar instead of binary units, however, approximate Boltzmann machines just as well as networks of units with Gaussian additive noise. The noise in these network is not state-dependent, but the model is biophysically questionable.

- In networks with asymmetric connectivity, the detailed-balance condition is not fulfilled in general and the ability to compute conditionals via constraints is lost.

Together these findings strongly suggest that the ability to sample from the exact conditionals by constraining the network dynamics via changes in the biases of individual units, is a particular feature of Boltzmann machines not easily replicated in general sampling networks. Further work is required to investigate in how far this impacts large-scale network models and practical applications.

Neural simulation technology

Modern neural simulation technology scales to the largest supercomputers available today (Kunkel et al., 2014). In chapter 8 we investigated the **limits of current neural simulation technology and its scalability to post-petascale systems**. We find that:

- For a purely postsynaptic storage of connection information, infrastructure overhead and MPI buffers will consume most of the available memory in systems larger than those currently available.
- To be able to support the simulation of networks at the scale of human cortex ($\sim 10^{10}$ neurons, $\sim 10^{14}$ synapses), simulators need to fully exploit the sparsity of these brain-scale networks and avoid global information stored on, and communicated to, each MPI rank.
- Introducing a two-tier connection infrastructure and directed MPI communication via `MPI_Alltoall` removes these scalability bottlenecks, leading to perfect weak scaling of memory usage.
- Implementing these changes in the NEST simulation kernel (Gewaltig and Diesmann, 2007), leads to significant performance improvements also on current HPC systems. For weak scaling, the new simulation kernel exhibits perfect scaling in terms of memory consumption on two of the largest systems available today, the JUQUEEN BlueGene/Q supercomputer at the Jülich Research Centre, Germany, and the K computer at the Advanced Institute for Computational Science in Kobe, Japan. Although our changes focused on memory-usage optimizations, the new kernel significantly reduces the wall-clock time for large-scale simulations: In our benchmarks we achieve a reduction by about 80% in a full-JUQUEEN simulation of a plastic network with slightly more than $5 \cdot 10^8$ neurons.

Our results demonstrate scalable simulation technology suitable for post-petascale HPC systems, that does not sacrifice flexibility, generality or performance in small-scale simulations.

With neural simulation technology developing at a rapid pace, researchers can construct complex neural network models, guided by functional considerations and biophysical plausibility. In chapter 9 we addressed a specific issue arising during the development of such models, namely how **functional network models in neural simulators can be provided with complex, non-stationary and reproducible stimuli**, instead of relying on ad-hoc implementations. The reliance on manually implemented environments and tasks for training and evaluating functional neural network models makes it difficult to compare models and is hence a bottleneck for progress in the domain of biophysical network models capable of learning and performing complex tasks. To tackle this issue:

- We presented a toolchain employing MUSIC (Djurfeldt et al., 2010b) and ZeroMQ (Hintjens, 2013) to communicate data between a running network simulation and environments from the OpenAI Gym (Brockman et al., 2016), thus providing researchers in computational neuroscience with a large variety of standardized benchmark tasks.
- We employed this toolchain to successfully train a rate-based reinforcement-learning network model derived from a spiking implementation (Frémaux, Sprekeler, and Gerstner, 2013) on two different environments.

While we specifically used the environments to train a reinforcement-learning agent, environments from the OpenAI Gym could in general be adapted to provide dynamic input to functional network models also in a purely passive, perception scenario. These can be used to evaluate models for probabilistic inference on a variety of tasks that are easily reproducible by other researchers. We hope that this toolchain helps to improve evaluation, comparison and reproducibility of biophysically plausible functional neural network models.

Chapter 11

Outlook

The Bayesian brain hypothesis is a promising candidate to provide us with an explicit, mature mathematical framework to understand dynamics and function in biological systems, and to create artificial agents closely mimicking intelligent organisms. To be successful, it needs to specify biophysically plausible mechanisms by which probability distributions are represented, learned from experience and combined across various sensory streams and between bottom-up and top-down inputs. In the recent decade, many proposals have been made to connect the behavioural with the neural level (Knill and Pouget, 2004; Fiser et al., 2010; Pouget et al., 2013; Haefner, Berkes, and Fiser, 2016), using spatial (Ma et al., 2006) as well as temporal representations of probability distributions in neural substrates (Buesing et al., 2011). Recently, typical signatures of sample-based representations have even been reported in electrophysiological data (Berkes et al., 2011; Orbán et al., 2016). To progress further, we need to develop mechanistic models on the cellular level with a higher degree of biological plausibility than previously, that are directly comparable to experimental data. These models need to take biological constraints into account, including the small trial-to-trial variability of responses in isolated neurons (Bryant and Segundo, 1976; Mainen and Sejnowski, 1995a), the unreliability of synaptic transmissions (Branco and Staras, 2009; Ribrault, Sekimoto, and Triller, 2011a) and the asymmetry of synaptic connections (Kandel, Schwartz, and Jessel, 2000). We further need to investigate in how far the relevant computations, for example computing conditionals from observed evidence or inverting conditionals via Bayes's rule, can be efficiently implemented given these constraints.

The extension to asymmetric, non-equilibrium networks will connect the sample-based view to the temporal domain. In recent work Aitchison and Lengyel (2016) have demonstrated how asymmetric networks performing a certain type of Monte-Carlo sampling can account for many dynamical features exhibited by neural networks in response to onset of a stationary stimulus. However, typically neural systems have to deal with non-stationary stimuli, and hence need to be able to represent directed temporal relations between subsequent states. The non-equilibrium dynamics exhibited by asymmetric networks allows an implicit representation of time by removing the constraint of detailed balance. Asymmetric networks are hence expected to possess higher computational power than their symmetric counterparts (Parisi, 1986; Apolloni and Falco, 1991; Li and Dayan, 1999). Currently, concrete proposals are missing of how the distributions underlying inference procedures with a temporal dimension are learned via biophysically plausible plasticity rules. Since for most natural tasks a supervisor is missing, the main focus should be on unsupervised learning that allows organisms to extract the statistical (temporal) structure of their environment. In addition, the extension of sample-based models to the temporal domain can be a first step to reconcile approximate-inference approaches with another prominent concept for brain function: reservoir computing

(Jaeger, 2001; Maass, Natschl ger, and Markram, 2002). In contrast to static computations predominantly considered in sample-based models, reservoir computing is mainly concerned with inherently temporal tasks, for example, learning and generation of motor trajectories (Sussillo and Abbott, 2009b; Hennequin, Vogels, and Gerstner, 2014). Currently, it is not clear, in how far these two approaches are related.

In pursuing the Bayesian brain hypothesis, our computational models need to reflect the finite computational power of the nervous system. As discussed in chapter 1, we need to be aware that biological neural networks most likely employ heuristics and shortcuts to probabilistic inference. In addition, as the nervous system is the result of natural evolution, we should only expect explicit representation of uncertainty, e.g., in the form of distributions, where it is of significant evolutionary advantage. For cases in which uncertainty plays a minor role a direct mapping from input to output might be sufficient for an organism and any search for representation of distributions in the corresponding neural circuitry would be in vein (cf. Ma, 2012). Seeking a close connection between computational models, biological implementations, and experimental data, we hence need to be undogmatic, keeping in mind that not every computation needs to be probabilistic and explicitly accounting in our models for the constraints in physical systems.

From an evolutionary perspective brains are one way to increase the fitness of a species by increasing the probability of individuals to survive through sophisticated information-processing capabilities (see, e.g., Pinker, 1999). In this thesis, we have focused on how the nervous system can efficiently gather information about the environment from uncertain sensory inputs and combine them with prior expectations, i.e., on probabilistic inference. While probability theory can solve the inference problem, the question remains of how organisms convert posterior distributions over states of the environment into an appropriate course of actions (see, e.g., Knill and Pouget, 2004; K rding, 2007). Up to now, we could avoid this discussion since the relation between inference and actions is fundamentally asymmetric: while actions require inferences, inferences do not necessarily require actions (Lindley, 2014).

Decision theory (Wald, 1939; Schlaifer and Raiffa, 1968; Jaynes, 2003a; Berger, 2013) provides a mathematical framework to determine an optimal course of actions for a given situation. To derive the optimal action for a particular agent, one defines a value function $V(y, \Omega)$ ¹ that assigns a value to an action y given the current state of the environment and internal state of the agent, collectively denoted by Ω . In the following we will also refer to Ω as the “world state”. In general, decision theory states that an agent chooses the action y^* that maximizes V given the current situation Ω (e.g., Jaynes, 2003a; K rding, 2007):

$$y^* = \arg \max_y V(y, \Omega) .$$

¹Accounting for the uncertainty in predicting the future, the value function $V(y, \Omega)$ can be interpreted as the expected value over all possible future world states Ω^+ that will be reached with probability $p(\Omega^+|y, \Omega)$:

$$V(y, \Omega) := \int d\Omega^+ W(\Omega^+) p(\Omega^+|y, \Omega) .$$

Here $W(\Omega^+)$ expresses the value of the future world state Ω^+ to the agent.

After defining a value function V one can hence determine the optimal course of actions. If we consider this as a model for brain function, it is unclear how exactly these value functions are learned, represented and evaluated. While we can hope for general inference procedures that are valid across a variety of species, value functions will most certainly differ: inferring the presence of a cat will typically lead to different courses of actions for mice or humans. Value functions are different across agents, change over the course of time and adapt to circumstances. One can argue that we already came across a similar problem when considering inference, where one needs to define a prior distribution, that depends on experience and on the current state of the agent. However, purely by passively observing a large number of states from the environment the prior expectation of an agent can be shaped such that the initial prior distribution has very little influence on subsequent inferences. In contrast, a value function can only be constructed through interaction with the environment and is hence much harder to learn from scratch, suggesting that value functions rely more on innate, i.e., genetically encoded knowledge. It is possible to gain some insights into the value function of humans by studying behavior in a wide range of circumstances and construct general value functions that can reproduce a large range of behaviors (e.g., Körding et al., 2004) and hence can be a reasonable starting point for computational models. Considering that the optimal course of actions heavily depends on the particular value function has significant consequences for constructing artificial agents: what value functions should they be equipped with and in how far should they be able to alter them?

Due to the weak dependence of the inference procedure and the strong dependence of the value function on the agent, it is conceptually useful to separate the two problems of perception and decision making. In a realistic setting the current world state Ω is not directly available to an organism, but needs to be inferred through sensory input X that is combined with prior expectations I . To determine the best action y^* , we hence need to average over all possible world states that are compatible with the sensory evidence and prior expectations. This allows us to explicitly separate the two parts:

$$y^* = \arg \max_y \int d\Omega \underbrace{v(y|\Omega)}_{\substack{\text{action} \\ \rightarrow \text{decision theory}}} \underbrace{p(\Omega|XI)}_{\substack{\text{perception} \\ \rightarrow \text{probabilistic inference}}} . \quad (11.1)$$

Since values V can in general be positive and negative, we use here an alternative representation of the value function by introducing $v \sim e^V$. Since the exponential function is monotonous this mapping conserves the position of extrema while allowing us to interpret v as a (conditional) probability density. Value distributions $v(y|\Omega)$ can hence be similarly represented and manipulated as distributions used in the inference part. If we consider implementations in biological substrates, however, it becomes unclear whether we can disentangle the action and perception part as suggested by equation (11.1) or whether we need to view them as an integrated whole. A priori, it is unclear why natural evolution should lead to systems in which these conceptually different parts are neatly separated. One can, however, hope that due to the hierarchical organization of cortex (Felleman and Van Essen, 1991; Markov et al., 2014c) a certain form of computation is prevalent in certain areas. While early sensory areas are most likely preoccupied with inference, this is probably less the case in motor areas and indeed, the probabilistic inference view has been very successful in particular describing perception (Hoyer and Hyvarinen, 2003; Berkes et al., 2011; Moreno-Bote, Knill, and Pouget, 2011; Orbán et al., 2016).

To disentangle the two parts and guide experimental approaches, computational circuit-level models can provide significant advantages, as they allow to identify specific signatures of inference and decision making that can be observed in experiments. This requires researchers on the one hand to develop models in which both parts, perception and action, play an important role and on the other hand an explicit incorporation of uncertainty in concrete tasks for functional network models and in experimental protocols.

Similar to behavioural models of probabilistic inference, we should not expect mathematically optimal behavior under any condition. Consequently we should consider a form of approximate decision theory, just as approximate inference, employing computational shortcuts to the full mathematical framework. This would mean that we should expect (close to) optimal decisions only under natural conditions (cf. Körding, 2007; Ma, 2012). If value functions can be represented as probability densities as suggested above, we do not need to reinvent the wheel to represent and compute with the corresponding values. Also for decision making, we could employ a sample-based approach: as we would expect strongly peaked posterior distributions for typical tasks, even within a sampling based picture, neural activity should show low variability, allowing for example for precise motor movement. The sample-based approach to approximate computations hence seems to provide a basis for the full spectrum of brain function, from inferring the state of the world to pursuing an appropriate course of actions.

Part IV

Appendices

Appendix A

Supplementary material to “Stochastic neural computation without noise”

A.1 Pairwise input correlations

Here we show how the covariance C_{kl}^{in} between the input fields of two units k and l receiving inputs from a pool of sources can be decomposed into a part arising from shared inputs and another from activity correlations. The input field for a single unit is given by equation (5.1) and hence:

$$\begin{aligned}
 C_{kl}^{\text{in}} &= \langle h_k h_l \rangle - \langle h_k \rangle \langle h_l \rangle \\
 &= \left\langle \left(\sum_i^K w_{ki} s_i + b_k \right) \left(\sum_j^K w_{lj} s_j + b_l \right) \right\rangle - \langle h_k \rangle \langle h_l \rangle \\
 &= \sum_i^K w_{ki} w_{li} A_i + \sum_i^K \sum_{j \neq i}^K w_{ki} w_{lj} C_{ij} \\
 &= C_{\text{shared},kl}^{\text{in}} + C_{\text{corr},kl}^{\text{in}}.
 \end{aligned}$$

We introduced the auto- and crosscovariances $A_i = \langle s_i^2 \rangle - \langle s_i \rangle^2$ and $C_{ij} = \langle s_i s_j \rangle - \langle s_i \rangle \langle s_j \rangle$ of the activities s_i and s_j of presynaptic neurons i and j , respectively. If Dale’s law is respected and the sign of all outgoing connections of a sources is unique, i.e. $\text{sign}(w_{ki}) = \text{sign}(w_{li})$, $\forall k, l, i$, the first term is always positive ($C_{\text{shared},kl}^{\text{in}} > 0$). For a pool of independently active presynaptic neurons, $C_{ij} = 0$ by definition and the second term in the input correlations hence vanishes ($C_{\text{corr},kl}^{\text{in}} = 0 \forall k, l$). The total input correlation is therefore always positive and determined by the number of shared sources. If the presynaptic sources are units in a recurrently connected network, their pairwise correlation is in general non-zero ($C_{ij} \neq 0$). In particular, in sparsely connected networks with sufficient inhibition, correlations arrange such that $C_{\text{corr},kl}^{\text{in}} \approx -C_{\text{shared},kl}^{\text{in}}$, leading to small remaining pairwise input correlations, $C_{kl}^{\text{in}} \approx 0$ (Renart et al., 2010b; Tetzlaff et al., 2012b).

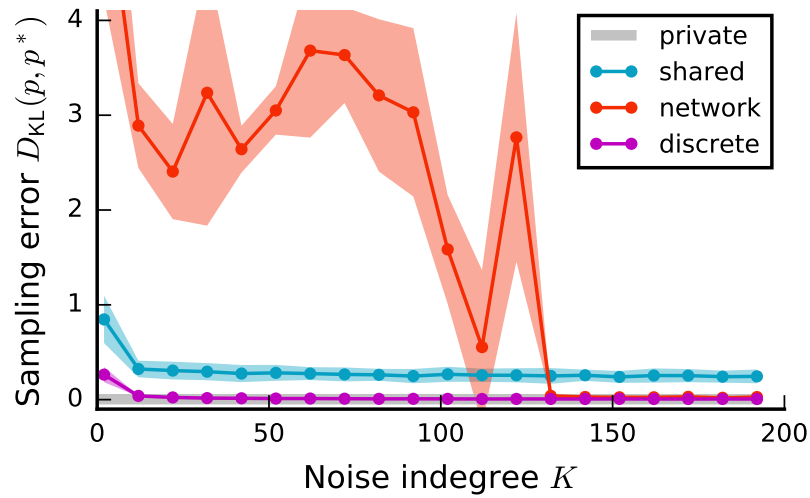


FIGURE A.1: Sampling error $D_{\text{KL}}(p, p^*)$ as a function of the number of background inputs K per sampling unit. Error bands indicate mean $\pm$ SEM over 5 random network realizations. Magenta (“discrete”) uses K separate sources for each sampling unit. Sampling duration $T = 10^5$ ms. Connectivity constant $K/N = 0.9$. Remaining parameters as in figure 5.4.

A.2 Sampling error depends on number of noise inputs per sampling unit

To closely approximate the effect of Gaussian noise on the input field, one needs a large number K of background inputs per sampling unit. Here, we scale the number of noise sources K per sampling unit, while also scaling the total number N of noise sources to keep their ratio constant. This allows us to investigate the impact of K without altering the amount of shared-input correlations. In addition to the three cases considered in the main manuscript (private, shared, network noise), we additionally consider the case of a separate pool of noise sources for each sampling unit (“discrete”), where shared-input correlations are absent.

For small K , the input distribution is strongly discretized and does not approximate Gaussian well, reflected in a large sampling error for very small K for the discrete and shared case (figure A.1). As we increase K , the sampling error decreases rapidly for the discrete case, and drops to the same level as Gaussian noise at about 50 inputs. For the shared case, the error decreases as well as we increase K , but is limited from below by sampling error introduced by shared-input correlations. For the network case, the sampling error is very large for small K as the network dynamics lock into a fixed point. However, for $K > 130$, the sampling error for the network case drops almost to the level of Gaussian noise.

A.3 Small, recurrent networks can supply large sampling networks with noise – no weight scaling

In figure 5.8 we scaled the weights in the sampling network with the size M of the sampling network as $1/\sqrt{M}$. Ignoring the influence of cross-correlations, this scaling keeps the variances of the input distribution arising from recurrent connections

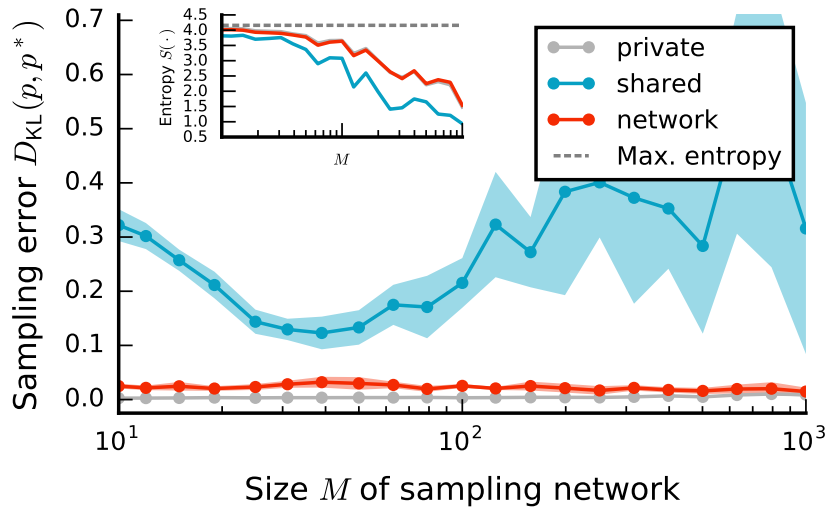


FIGURE A.2: Same as figure 5.8, but with constant average weight in sampling networks: $\mu_{\text{BM}} = -0.15$.

in the sampling network constant. Effectively this leads to approximately constant entropy for a large range of sampling network sizes.

If we do not scale the weights as above when increasing the size of the sampling network, the input variance increases and the relative noise strength hence decreases, leading to an effectively stronger coupled sampling network. This strongly decreases the entropy of the sampled distribution (figure A.2, inset). Despite the decrease in entropy, the sampling errors for the private and network cases stay approximately constant (figure A.2). For the shared case, the sampling error initially decreases due to the strengthened effective feedback that suppresses shared-input correlations arising from the limited pool of background sources (cf. **Small, recurrent networks can supply large sampling networks with noise**). As the size of the sampling network increases the sampling error increases again from about $M = 40$. This is most likely caused by the decrease in the relative noise strength and the sampling dynamics hence becoming too slow to approximate the target distribution in the finite sampling duration considered here.

A.4 Simulation details

table A.1, A.2, A.3, A.4, A.5 summarize the binary network model and parameters.

table A.6, A.7, A.8, A.9 summarize the spiking network model and parameters. Simulations carried out with NEST 2.10 (Bos et al., 2015).

A		Model summary	
Populations	One		
Topology	—		
Connectivity	All-to-all		
Neuron model	Stochastic binary units		
Channel models	—		
Synapse model	—		
Plasticity	—		
External input	—		
Measurements	Binary states of m units		
B		Populations	
Name	Elements	Size	
Sampling network	Stoch. binary units	M	
C		Connectivity	
Source	Target	Pattern	
Sampling network	Sampling network	All-to-all, random weights drawn from Beta distribution, $w_{ij} \sim \text{Beta}(a, b)$, symmetric connections $w_{ij} = w_{ji}$, no self connections $w_{ii} = 0$	
D		Neuron model	
Type	Stochastic binary units		
Dynamics	Transition into state 1 according to probability determined by the activation function $F_i(h_i) = \frac{1}{1+e^{-\beta h_i}}$ with input field $h_i = \sum_j w_{ij} s_j + b_i$.		

TABLE A.1: Description of the sampling network model with intrinsic noise (according to Nordlie, Gewaltig, and Plesser (2009a)).

A		Model summary	
Populations	One		
Topology	—		
Connectivity	All-to-all		
Neuron model	Stochastic binary units		
Channel models	—		
Synapse model	—		
Plasticity	—		
External input	—		
Measurements	Binary states of m units		
B		Populations	
Name	Elements	Size	
Sampling network	Stoch. binary units	M	
C		Connectivity	
Source	Target	Pattern	
Sampling network	Sampling network	All-to-all, random weights drawn from Beta distribution, $w_{ij} \sim \text{Beta}(a, b)$, symmetric connections $w_{ij} = w_{ji}$, no self connections $w_{ii} = 0$	
D		Neuron model	
Type	Stochastic binary units		
Dynamics	Transition into state 1 according to probability determined by the activation function $F_i(h_i) = \frac{1}{2}\text{erfc}\left(\frac{h_i + \mu_i}{\sqrt{2\sigma^2}}\right)$ with input field $h_i = \sum_j w_{ij}s_j + b_i$.		

TABLE A.2: Description of sampling network model with private noise (according to Nordlie, Gewaltig, and Plesser (2009a)).

A		Model summary	
Populations	Three		
Topology	—		
Connectivity	All-to-all; sparse random with fixed indegree		
Neuron model	Stochastic binary units, deterministic binary units		
Channel models	—		
Synapse model	—		
Plasticity	—		
External input	—		
Measurements	Binary states		
B			
Populations			
Name		Elements	Size
Sampling network		Det. binary units	M
Background pop. (E)		Stoch. binary units	γN
Background pop. (I)		Stoch. binary units	$(1 - \gamma)N$
C			
Connectivity			
Source		Target	Pattern
Sampling network		Sampling network	All-to-all, random weights drawn from Beta distribution, $w_{ij} \sim \text{Beta}(a, b)$, symmetric connections $w_{ij} = w_{ji}$, no self connections $w_{ii} = 0$
Background pop. (E)		Sampling network	Random convergent $\gamma K \rightarrow 1$, weight w
Background pop. (I)		Sampling network	Random convergent $(1 - \gamma)K \rightarrow 1$, weight $-gw$
D			
Neuron model			
Type		Stochastic binary units	
Dynamics		Transition into state 1 according to probability determined by the activation function $F_i(h_i) = \frac{1}{1+e^{-\beta h_i}}$ with input field $h_i = \sum_j w_{ij}s_j + b_i$.	
Type		Deterministic binary units	
Dynamics		Transition into state 1 according to the activation function $F_i(h_i) = \Theta(h_i)$ with input field $h_i = \sum_j w_{ij}s_j + b_i$.	
E			
Measurements			
Binary states of m units from sampling network			

TABLE A.3: Description of sampling network model with shared noise (according to Nordlie, Gewaltig, and Plesser (2009a)).

A			Model summary		
Populations		Three			
Topology		—			
Connectivity		All-to-all; sparse random with fixed indegree			
Neuron model		Deterministic binary units			
Channel models		—			
Synapse model		—			
Plasticity		—			
External input		—			
Measurements		Binary states			
B			Populations		
Name		Elements		Size	
Sampling network		Det. binary units		M	
Background pop. (E)		Det. binary units		γN	
Background pop. (I)		Det. binary units		$(1 - \gamma)N$	
C			Connectivity		
Source		Target		Pattern	
Sampling network		Sampling network		All-to-all, random weights drawn from Beta distribution, $w_{ij} \sim \text{Beta}(a, b)$, symmetric connections $w_{ij} = w_{ji}$, no self connections $w_{ii} = 0$	
Background pop. (E)		Sampling network		Random convergent $\gamma K \rightarrow 1$, weight w	
Background pop. (I)		Sampling network		Random convergent $(1 - \gamma)K \rightarrow 1$, weight $-gw$	
Background pop. (E)		Background pop. (E)		Random convergent $\gamma K \rightarrow 1$, weight w	
Background pop. (E)		Background pop. (I)		Random convergent $\gamma K \rightarrow 1$, weight w	
Background pop. (I)		Background pop. (E)		Random convergent $(1 - \gamma)K \rightarrow 1$, weight $-gw$	
Background pop. (I)		Background pop. (I)		Random convergent $(1 - \gamma)K \rightarrow 1$, weight $-gw$	
D			Neuron model		
Type		Deterministic binary units			
Dynamics		Transition into state 1 according to the activation function $F_i(h_i) = \Theta(h_i)$ with input field $h_i = \sum_j w_{ij}s_j + b_i$.			
E			Measurements		
Binary states of m units from sampling network					

TABLE A.4: Description of sampling network model with network noise (according to Nordlie, Gewaltig, and Plesser (2009a)).

B		Populations	
Name	Values		
M	100*		
N	222*		
γ	0.3		
C		Connectivity	
Name	Values		
a	2		
b	2		
K	200		
w	0.3		
g	8		
D		Neuron model	
Name	Values		
β	1*		
μ	0		
σ	from β via equation (5.9)		
E		Measurements	
Name	Values		
m	6		
Miscellaneous			
Name	Values	Description	
$\bar{s}$	0.4	Average activity in sampling networks	
$\bar{z}$	0.3	Average activity in background population	
T_{sim}	10^5 ms*	Simulation time	
T_{warmup}	500 ms	Warmup time (ignored during analysis)	
τ	10 ms	Average inter-update interval	

TABLE A.5: Parameters for binary network simulations (according to Nordlie, Gewaltig, and Plesser (2009a)). Stars indicate default values.

A		Model summary	
Populations		One	
Topology		—	
Connectivity		All-to-all	
Neuron model		Leaky integrate-and-fire (LIF)	
Channel models		—	
Synapse model		Exponentially decaying currents, fixed delays	
Plasticity		—	
External input		Poisson-distributed spike trains	
Measurements		Spikes	
B		Populations	
Name	Elements	Size	
Sampling network	LIF neuron	M	
C		Connectivity	
Source	Target	Pattern	
Sampling network	Sampling network	All-to-all, random weights drawn from Beta distribution, $w_{ij} \sim \text{Beta}(a, b)$, symmetric connections $w_{ij} = w_{ji}$, no self connections $w_{ii} = 0$, translation from binary-unit domain to spiking neurons via constant calibration factors (see section 5.2.9)	
D		Neuron and synapse model	
Type	Leaky integrate-and-fire, exponential currents		
Subthreshold dynamics	Subthreshold dynamics ($t \notin (t^*, t^* + \tau_{\text{ref}})$): $C_m \frac{\text{d}}{\text{d}t} v(t) = -g_l(v(t) - E_l) + I_{\text{syn}}(t)$ Reset and refractoriness ($t \in (t^*, t^* + \tau_{\text{ref}})$): $v(t) = v_{\text{reset}}$		
Current dynamics	$\tau_{\text{syn}} \frac{\text{d}}{\text{d}t} I_{\text{syn}}(t) = -I_{\text{syn}}(t) + \sum_{i,k} J \delta(t - t_i^k)$ Here the sum over i runs over all presynaptic neurons and the sum over k over all spike times of the respective neuron i		
Spiking	If $v(t^* -) < \Theta \wedge v(t^* +) \geq \Theta$: emit spike with time stamp t^*		
E		Measurements	
Spike trains recorded from m neurons from the sampling network			
F		External input	
Per neuron, one private excitatory and one inhibitory Poisson source with rate ν_{ex} and ν_{in} , respectively.			

TABLE A.6: Description of spiking sampling network model with private noise (according to Nordlie, Gewaltig, and Plesser (2009a)).

A		Model summary	
Populations	One		
Topology	—		
Connectivity	All-to-all; sparse random with fixed indegree		
Neuron model	Leaky integrate-and-fire (LIF)		
Channel models	—		
Synapse model	Exponentially decaying currents, fixed delays		
Plasticity	—		
External input	Poisson-distributed spike trains		
Measurements	Spikes		
B		Populations	
Name	Elements	Size	
Sampling network	LIF neuron	M	
C		Connectivity	
Source	Target	Pattern	
Sampling network	Sampling network	All-to-all, random weights drawn from Beta distribution, $w_{ij} \sim \text{Beta}(a, b)$, symmetric connections $w_{ij} = w_{ji}$, no self connections $w_{ii} = 0$, translation from binary-unit domain to spiking neurons via constant calibration factors (see section 5.2.9)	
D		Neuron and synapse model	
See table A.6.			
E		Measurements	
See table A.6.			
F		External input	
Per neuron, γK excitatory and $(1 - \gamma)K$ inhibitory Poisson sources with weight J , rate $\tilde{\nu}_{\text{ex}}$ and weight $-gJ$, rate $\tilde{\nu}_{\text{in}}$, respectively. Excitatory and inhibitory inputs randomly chosen from a common pool of γN and $(1 - \gamma)N$ units, respectively.			

TABLE A.7: Description of spiking sampling network model with shared noise (according to Nordlie, Gewaltig, and Plesser (2009a)).

A			Model summary		
Populations		Three			
Topology		—			
Connectivity		All-to-all; sparse random with fixed indegree			
Neuron model		Leaky integrate-and-fire (LIF)			
Channel models		—			
Synapse model		Exponentially decaying currents, fixed delays			
Plasticity		—			
External input		Resting potential above firing threshold in background populations			
Measurements		Spikes			
B			Populations		
Name		Elements	Size		
Sampling network		LIF neuron	M		
Background pop. (E)		LIF neuron	γN		
Background pop. (I)		LIF neuron	$(1 - \gamma)N$		
C			Connectivity		
Source		Target	Pattern		
Sampling network		Sampling network	All-to-all, random weights drawn from Beta distribution, $w_{ij} \sim \text{Beta}(a, b)$, symmetric connections $w_{ij} = w_{ji}$, no self connections $w_{ii} = 0$, translation from binary-unit domain to spiking neurons via constant calibration factors (see section 5.2.9)		
Background pop. (E)		Sampling network	Random convergent, $\gamma K \rightarrow 1$, weight w , delay d		
Background pop. (I)		Sampling network	Random convergent, $(1 - \gamma)K \rightarrow 1$, weight $-gw$, delay d		
Background pop. (E)		Background pop. (E)	Random convergent, $\gamma K \rightarrow 1$, weight w , delay d		
Background pop. (E)		Background pop. (I)	Random convergent, $\gamma K \rightarrow 1$, weight w , delay d		
Background pop. (I)		Background pop. (E)	Random convergent, $(1 - \gamma)K \rightarrow 1$, weight $-gw$, delay d		
Background pop. (I)		Background pop. (I)	Random convergent, $(1 - \gamma)K \rightarrow 1$, weight $-gw$, delay d		
D			Neuron and synapse model		
See table A.6.					
E			Measurements		
See table A.6.					

TABLE A.8: Description of spiking sampling network model with network noise (according to Nordlie, Gewaltig, and Plesser (2009a)).

B		Populations	
See table A.5.			
C		Connectivity	
Name	Values		
a	2		
b	2		
K	1000		
J	0.002 nA (0.0635 nA)		
g	2		
D		Neuron model	
Name	Values		
τ_{ref}	10.0 ms (0.1 ms)		
τ_{syn}	10.0 ms (5.0 ms)		
C_{m}	0.2 nF (1.0 nF)		
g_{l}	$2.0\text{ }\mu\text{S}$ ($0.05\text{ }\mu\text{S}$)		
E_{l}	-50.00 mV (-40.00 mV)		
v_{reset}	-50.01 mV (-60.00 mV)		
Θ	-50.00 mV		
E		Measurements	
Name	Values		
m	10		
Miscellaneous			
Name	Values	Description	
T_{sim}	10^7 ms	Simulation time	
T_{warmup}	10^3 ms	Warmup time (ignored during analysis)	
F		External input	
Name	Values		
ν_{ex}	10 kHz		
ν_{in}	10 kHz		
$\tilde{\nu}_{\text{ex}}$	$4.4 \pm 0.1\text{ Hz}$		
$\tilde{\nu}_{\text{in}}$	$4.4 \pm 0.1\text{ Hz}$		

TABLE A.9: Table of parameters for spiking network simulations (according to Nordlie, Gewaltig, and Plesser (2009a)). Values without parantheses are for the sampling network, values in parantheses for the noise network.

Appendix B

Supplementary material to “The effect of heterogeneity on decorrelation mechanisms in spiking neural networks: a neuromorphic-hardware study”

B.1 Network description

Details on the network model as well as parameter values are provided in table B.1 and B.2, respectively.

B.2 Description of data analysis

A summary of the quantities and the data analysis used in this study is provided in table B.3. Parameter values for the data analysis are given in table B.4.

B.3 Power consumption

The *Spikey* system consumes approximately 6 W of power, and the chip itself less than 0.6 W. On the chip most power is consumed by digital communication infrastructure, which is not part of the neuromorphic network. In the following, we estimate the power consumption for a single synaptic event using the data set partly shown in figure 6.5a. This emulation lasts $T = 10$ s in biological time and generates approximately $45 \cdot 10^3$ spikes. Considering the acceleration of the hardware network (10^4) and the synapse count per neuron ($K = 15$), the system generates $7 \cdot 10^8$ synaptic events per second in hardware time. If we consider the total power consumption of the *Spikey* chip, the upper bound of energy consumed by each synaptic transmission will be approximately 1 nJ. Because these measurements include the communication infrastructure and other support electronics to observe spike times and membrane traces, the real energy consumption for synaptic transmissions is estimated to be approximately ten times smaller. Network simulations on conventional supercomputers a far less energy efficient and consume tens of μ J for each synaptic transmission (Sharp et al., 2012).

A		Model summary	
Populations		One (inhibitory)	
Topology		—	
Connectivity		Random convergent connections (fixed in-degree)	
Neuron model		Leaky integrate-and-fire (LIF), fixed firing threshold, fixed absolute refractory time	
Channel models		—	
Synapse model		Exponentially decaying conductances, fixed delays	
Plasticity		—	
External input		Resting potential higher than threshold (= constant current) ($E_l > \Theta$)	
Measurements		Spikes and membrane potentials from all neurons	
Other		No autapses, no multapses	
B		Populations	
Name	Elements	Size	
I	LIF neuron	N	
C		Connectivity	
Source	Target	Pattern	
I	I	Random convergent connections, in-degree K	
D		Neuron and synapse model	
Type	Leaky integrate-and-fire, exponentially decaying conductances		
Subthreshold dynamics	Subthreshold dynamics ($t \notin (t^*, t^* + \tau_{\text{ref}})$): $C_m \frac{d}{dt} v(t) = -g_l(v(t) - E_l) - g_{\text{syn}}(t)(v(t) - E_{\text{inh}})$ Reset and refractoriness ($t \in (t^*, t^* + \tau_{\text{ref}})$): $v(t) = v_{\text{reset}}$ This model is emulated by analog circuitry on the <i>Spikey</i> chip (Indiveri et al., 2011a).		
Conductance dynamics	For each presynaptic spike at time t^* ($t > t^* + d$): $g_{\text{syn}}(t) \approx J \exp(-\frac{t-t^*-d}{\tau_{\text{syn}}})\Theta(t)$, with $J = w_{\text{hw}}g_{\text{max}}$ and Heaviside function $\Theta(t)$. This model is emulated by analog circuitry on the <i>Spikey</i> chip (Pfeil et al., 2013b).		
Spiking	If $v(t^*-) < \Theta \wedge v(t^*+) \geq \Theta$: emit spike with time stamp t^*		

TABLE B.1: Description of the network model (according to Nordlie, Gewaltig, and Plesser (2009b)).

B Populations		
Name	Values	Description
N	{96, 112, 128, 144, 160, 176, 192 }	network size
C Connectivity		
Name	Values	Description
K	15	number of presynaptic partners
D Neuron		
Name	Values	Description
v_{reset}	-80 mV	reset potential
E_l	-52 mV	resting potential
Θ	-62 mV	firing threshold
E_{inh}	-80 mV	inhibitory reversal potential
τ_{ref}	1 ms	refractory period
C_m	0.2 nF	membrane capacitance
g_l	<i>tuned during calibration process</i>	leak conductance
τ_m^{eff}	uncalibrated: 10.50(5.94, 17.90)ms calibrated: 10.77(6.22, 18.30)ms	effective membrane time constant
D Synapse		
Name	Values	Description
g_{max}	in the order of 1 nS	conductance amplitude
w_{hw}	3	synaptic weight (in hardware values $\in [0, 15]$)
$V_{\text{max}}^{\text{eff}}$	uncalibrated: -5.57(-12.50, -2.62)mV calibrated: -6.05(-14.37, -2.92)mV	effective post-synaptic potential prefactor
$\tau_{\text{syn}}^{\text{eff}}$	uncalibrated: 3.78(2.52, 5.58)ms calibrated: 4.17(2.76, 6.18)ms	effective synaptic-current time constant
d^{eff}	uncalibrated: 2.16(1.91, 2.48)ms calibrated: 2.26(1.93, 2.55)	effective synaptic delay
Other Software		
Name	Values	Description
SpikeyHAL	9e86d11c	git revision
PyNN	2fe40b43	git revision
vmodule	76ef3b44	git revision
logger	826c5ed6	git revision
Other Hardware		
Name	Values	Description
	chip 508 (version 5)	chip used for main manuscript
	chip 503, 504 (version 5) and 603, 605, 666 (version 4)	chips used for supplements

TABLE B.2: Parameter values for the network model described in table B.1. Bold numbers indicate default values. Gray numbers indicate target values not considering fixed-pattern noise. Leak conductances g_l are adjusted in the calibration process (see section 6.2.5). τ_m^{eff} , $\tau_{\text{syn}}^{\text{eff}}$, d^{eff} and $V_{\text{max}}^{\text{eff}}$ describe effective values measured from spike-triggered averages in RAND emulations as described in appendix B.12. Effective values denote the median across synapses and trials followed by the 25th and 75th percentiles in brackets.

A		Analysis measures
Measure		Details
spike density		$\xi_i(t) = \sum_k \delta(t - t_i^k)$
spike train		$s_i(t_k) =$ number of spikes of neuron i per bin $[k\Delta t, (k+1)\Delta t)$
population activity		$\bar{s}(t) = \frac{1}{N} \sum_i s_i(t)$
time-averaged population activity		$\bar{\nu} = \langle \bar{s}(t) \rangle_t$
membrane potential		$v_i(t_k) =$ membrane potential of neuron i in bin $[k\Delta t, (k+1)\Delta t)$
(finite time) Fourier transform		$X_i(f) = \mathfrak{F}[x_i(t)](f) = \int_0^T dt x_i(t) e^{-2\pi i f t}$ (with inverse $\mathfrak{F}^{-1}$)
(single unit) power spectrum		$A_i(f) = \frac{1}{T} X_i^*(f) X_i(f)$
population-averaged power spectrum		$A(f) = \frac{1}{N} \sum_i A_i(f)$
population power spectrum		$\bar{A}(f) = \frac{1}{T} (\sum_i S_i^*(f)) (\sum_j S_j(f))$
pairwise cross spectrum		$C_{ij} = \frac{1}{T} X_i^*(f) X_j(f), i \neq j$
population-averaged cross spectrum		$C(f) = \frac{1}{N(N-1)} \sum_{i \neq j} C_{ij}(f) \equiv \frac{1}{N(N-1)} (\bar{A}(f) - N A(f))$ (note: $C(f) \in \mathbb{R}$)
sliding window filter		$X(f) \rightarrow X(f) * H(f)$ with $H(f) = \frac{1}{f_1 - f_0} \Theta(f - f_0) \Theta(f_1 - f)$
coherence		$\kappa(f) = \frac{C(f)}{A(f)}$
low-frequency coherence		$\kappa_X = \frac{1}{f_{\max} - f_{\min}} \int_{f_{\min}}^{f_{\max}} df \kappa(f)$
pop.-averaged cross-correlation function		$c(\tau) = \frac{1}{N(N-1)} \sum_{i \neq j} \langle s_i(t) s_j(t + \tau) \rangle_t \equiv \mathfrak{F}^{-1}[C(f)](\tau)$
time average		$\langle \dots \rangle_t$

TABLE B.3: Summary of the data analysis. Here $i \in [1, N]$, $X \in S, V$.

A			Analysis parameters
Parameter	Description	Values	
Δt	bin size for spike trains	1 ms	
Δt_m	bin size for membrane potential traces	0.52 ms	
T_{warmup}	initial warm-up time (not considered in analysis)	1 s	
T	emulated network time (biol. time domain)	10 s	
M	number of network realizations	{50, 100 }	
ΔF	width of sliding window	1 Hz	
$f_{\min}, f_{\max}$	interval boundaries for low-frequency coherence	0.1 Hz, 20 Hz	
a	calibration state	{0, $\frac{1}{8}$, $\frac{2}{8}$, $\frac{3}{8}$, $\frac{4}{8}$, $\frac{5}{8}$, $\frac{6}{8}$, $\frac{7}{8}$, 1 }	

TABLE B.4: Summary of analysis parameters (default values in bold).

B.4 Modification of the bisection method

In each iteration of the bisection method that is used to calibrate the leak conductances of hardware neurons (section 6.2.5), we evaluated the firing rate for each neuron by the median over $L = 25$ identical trials. However, if this measure is compared between consecutive identical iterations, temporal noise on time scales longer than the duration of one iteration may still lead to variability. In the original bisection method, the interval of possible solutions is halved after each iteration step (Press et al., 2007). To improve the convergence of this method in the context of our calibration we expanded the halved interval by 20% at both ends after each iteration. This prevents the algorithm to get stuck in an interval wrongly chosen by random fluctuations of the firing rates.

B.5 Effective weights

We quantify the effect of a single spike of neuron j on the firing rate of a postsynaptic neuron i by the effective weight w_{ij} of the connection $i \leftarrow j$. Assuming that the activity of neuron i does not affect the activity of neuron j (i.e., the RAND case), we define w_{ij} as the cross-correlation between the spike trains $s_j(t)$ and $s_i(t + \tau)$, where, due to causality, τ is positive. Then, we average the effective weight over the emulated network time T , and subtract the baseline determined by the average correlation for negative τ :

$$w_{ij} = \frac{1}{\tau_{\max}} \int_0^{\tau_{\max}} \langle s_j(t) s_i(t + \tau) \rangle_t d\tau - \frac{1}{\tau_{\min}} \int_{-\tau_{\min}}^0 \langle s_j(t) s_i(t + \tau) \rangle_t d\tau \quad . \quad (\text{B.1})$$

Here, we chose $\tau_{\max} = 50$ ms and $\tau_{\min} = -50$ ms. $\tau_{\max}$ was determined by measuring the average duration in which a spike from neuron j has an influence on neuron i (data not shown). $\tau_{\min}$ was then chosen symmetrically. The mass of the effective weights density shifts towards less negative effective weights for increasing heterogeneity (appendix figure B.1a).

We obtain $\bar{w}$ by averaging over all possible connections:

$$\bar{w} = \frac{1}{NK} \sum_{i,j} w_{ij} \quad . \quad (\text{B.2})$$

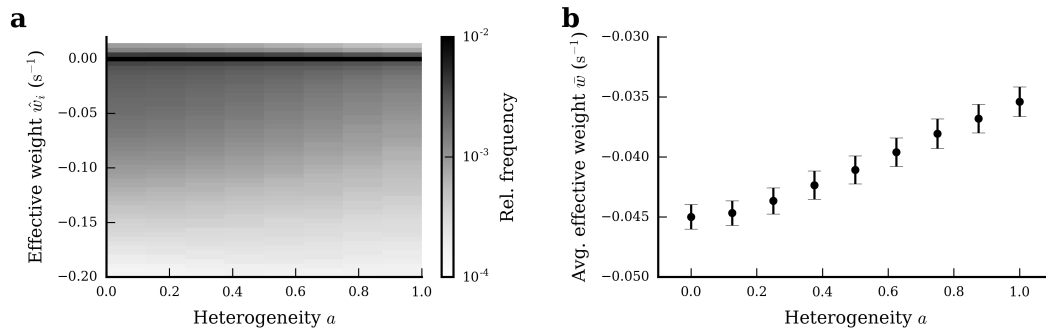


FIGURE B.1: Distribution of effective weights w (a; log scale) and population-averaged effective weights $\bar{w}$ (b) for different levels of heterogeneity, averaged over $M = 100$ network realizations. Symbols and error bars denote mean and one standard deviation, respectively.

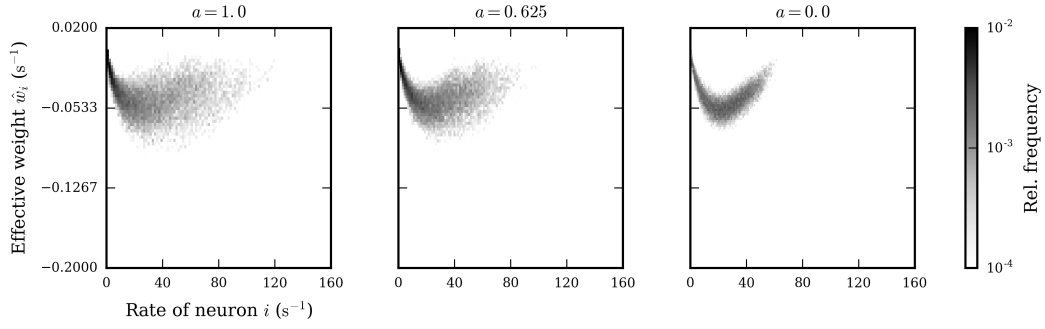


FIGURE B.2: Distribution of average incoming effective weights $\hat{w}_i$ over the rates of the postsynaptic neurons i , for three different levels of heterogeneity ($a = 0$, $a = 0.625$ and $a = 1$), averaged over $M = 100$ network realizations. The average incoming effective weights are obtained by summing over all presynaptic partners ($\hat{w}_i = \sum_j w_{ij}$).

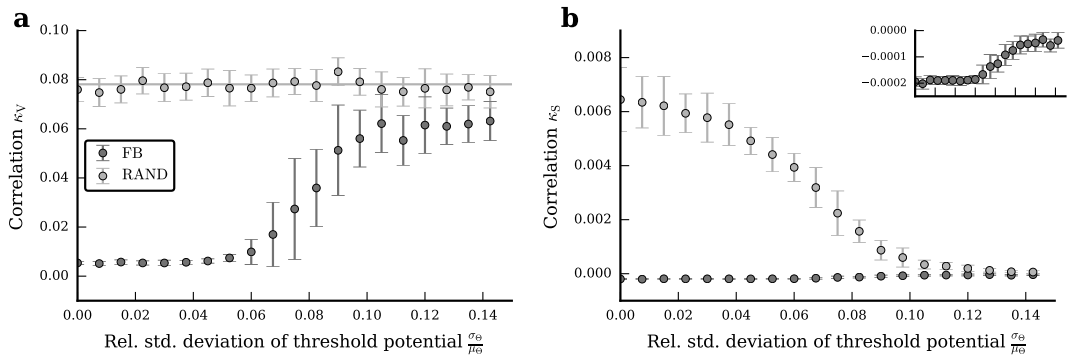


FIGURE B.3: Like figure 6.9, but for $N = 4800$ neurons and $K = 375$ inputs for each neuron. Note the different scales of the ordinate in figure 6.9 and (b). Symbols and error bars denote mean and standard deviation, respectively, across $M = 20$ network realizations (error bars are partly covered by markers).

For increasing heterogeneity the absolute value of the average effective weight $\bar{w}$ decreases (appendix figure B.1b). This can be explained by the dependence of the effective weight on the firing rate of the postsynaptic neuron (appendix figure B.2). In the regime of small rates ($< 10 \text{ s}^{-1}$), incoming spikes hardly affect the neuron's firing and the effective weight is small. Similarly, in the case for large rates ($> 40 \text{ s}^{-1}$). Neurons with intermediate firing rates are sensitive to input and hence have a more negative effective weight. Heterogeneity increases the number of neurons with small and large firing rates, and hence the absolute value of the average effective weight decreases, which in turn weakens the effective negative feedback of the network.

B.6 Simulations with software

We validate our results by comparing them to simulations with software (NEST Gewaltig and Diesmann (2007), PyNN Davison et al. (2008b)). In these simulations we modulated the degree of heterogeneity by distributing the firing thresholds of all neurons according to a normal distribution with mean Θ and standard deviation σ_Θ . Details about the network, neuron and synapse models and their parameters can be found in appendix table B.5 and B.6, respectively. The results are qualitatively similar to network emulations on the *Spikey* chip (compare figure 6.8 to 6.9) and also

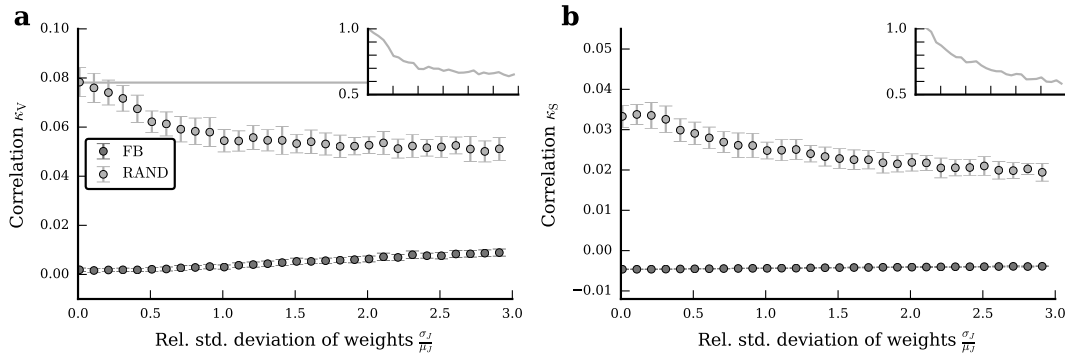


FIGURE B.4: Dependence of population-averaged input correlations **(a)**, and spike-train correlations **(b)** on the width of the weight distribution, for the intact network (FB, dark gray circles) and the RAND (light gray circles) case. Symbols and error bars denote mean and standard deviation, respectively, across $M = 30$ network realizations (error bars are partly covered by markers). The gray line in (a) depicts shared-input correlations in a homogeneous network (equation (6.2)). The insets show correlations normalized to unity at $\sigma_J = 0$, with the same abscissa as in the main plot. Note that in simulations the $\text{FB}_{\text{replay}}$ is identical to the FB case, and is hence not shown. Networks simulated with NEST (Gewaltig and Diesmann, 2007) and PyNN (Davison et al., 2008b).

hold for larger network sizes (appendix figure B.3). In the FB case, input correlations and output correlations increase with network heterogeneity. In contrast to hardware emulations, input correlations stay approximately constant in the RAND case. Output correlations strongly decrease with the standard deviation σ_Θ . Here, heterogeneity only affects the output spike times, and not the integrative properties of the neurons (see also section 6.3.2 and Yim, Aertsen, and Rotter (2013)).

In addition, we compare the effect of heterogeneity in firing thresholds on correlations to that of heterogeneity in synaptic weights. Details about the network, neuron and synapse models and their parameters can be found in appendix table B.5 and B.7, respectively. If we distribute synaptic weights, shared-input correlations decrease with the width of the weight distribution (appendix figure B.4). Output correlations are also reduced, however, only proportionally to the reduction of input correlations (compare insets in appendix figure B.4). While output correlations are overall smaller than input correlations due to the non-linearity in spike generation, we do not observe a boost of this decrease for large heterogeneities. Overall, the dynamics of the recurrent system is more sensitive to heterogeneities in firing thresholds than in synaptic weights (compare scale of abscissas of figure 6.9 to appendix figure B.4).

To investigate the generalization of our results to mixed excitatory-inhibitory networks, correlations were measured in a network of $N = 192$ neurons consisting of half excitatory and half inhibitory neurons. Details about the network, neuron and synapse models and their parameters can be found in appendix table B.8 and B.9, respectively. As described above, we distribute the firing thresholds of all neurons according to a normal distribution. The results are consistent with those obtained from purely inhibitory networks on hardware demonstrating the generality of our findings (compare figure 6.8 to appendix figure B.5). By modulating the level of heterogeneity separately for the excitatory or the inhibitory population, we observe that network dynamics are more sensitive to heterogeneities in the inhibitory than in the excitatory population (data not shown).

We investigate the effect of temporal noise on correlations by dithering spikes

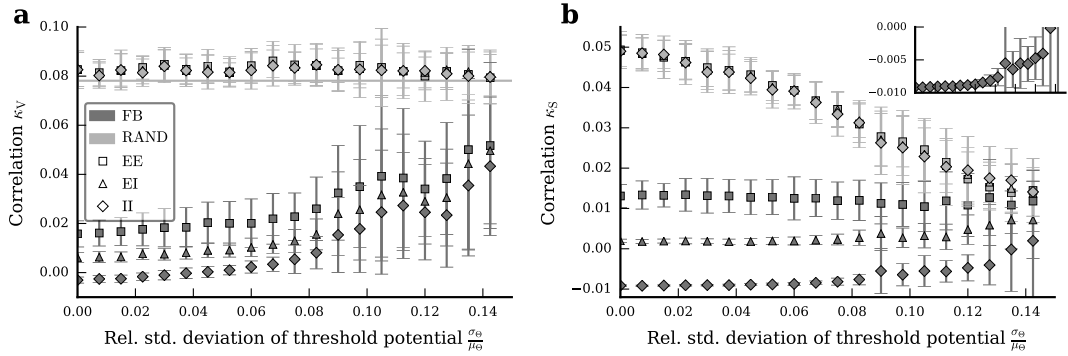


FIGURE B.5: Dependence of population-averaged input correlations **(a)**, and spike-train correlations **(b)** on the width of the threshold distribution, for the intact network (FB, dark gray circles) and the RAND (light gray circles) case. The network consists of two populations, one excitatory and one inhibitory, of equal size. Symbols and error bars denote mean and standard deviation, respectively, across $M = 30$ network realizations (error bars are partly covered by markers). The gray line in (a) depicts shared-input correlations in a homogeneous network (equation (6.2)). The inset in (b) shows a magnified view of the spike-train correlations in the FB case. Note that in simulations the $\text{FB}_{\text{replay}}$ is identical to the FB case, and is hence not shown. Networks are simulated with NEST (Gewaltig and Diesmann, 2007) and PyNN (Davison et al., 2008b).

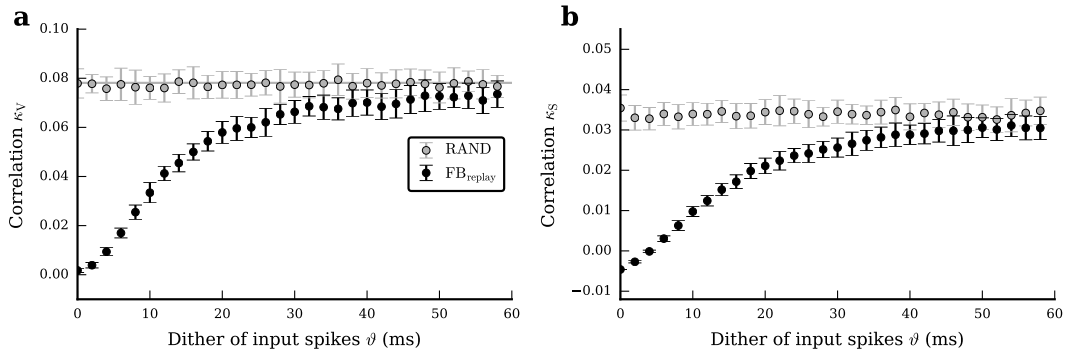


FIGURE B.6: Dependence of population-averaged input correlations **(a)**, and spike-train correlations **(b)** on the strength of temporal noise, for the $\text{FB}_{\text{replay}}$ (black circles) and the RAND (light gray circles) case. Symbols and error bars denote mean and standard deviation, respectively, across $M = 30$ network realizations (error bars are partly covered by markers). Gray curve in (a) depicts shared-input correlations in a homogeneous network (equation (6.2)). Correlations in the FB case correspond to correlations in the $\text{FB}_{\text{replay}}$ case for $\vartheta = 0$ ms. Networks simulated with NEST (Gewaltig and Diesmann, 2007) and PyNN (Davison et al., 2008b).

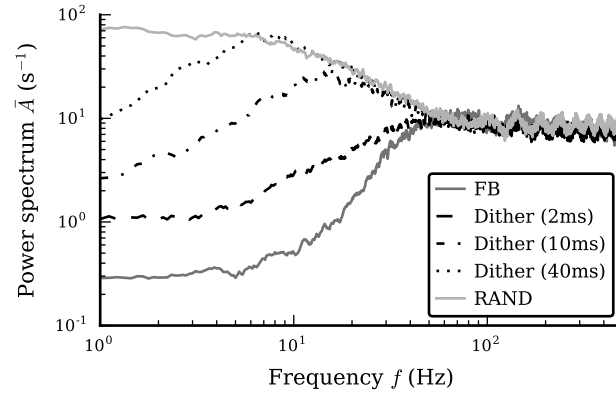


FIGURE B.7: Power spectrum of the population activity of the networks in appendix figure B.6 for the intact network (solid dark gray), different degrees of dither on input spikes (dashed, dashed dotted and dotted black) and the RAND case (solid light gray), averaged across $M = 30$ network realizations.

in the $\text{FB}_{\text{replay}}$ case before replaying them to the network (for network, neuron and synapse models see appendix table B.5 and B.6). Each spike time t is replaced by a spike time t' randomly drawn from a normal distribution with width ϑ : $t' \sim \mathcal{N}(t, \vartheta^2)$. Even for small ϑ , correlations increase on the input as well as on the output side, demonstrating the sensitivity of correlations to perturbations in the feedback loop (see appendix figure B.6 and Tetzlaff et al. (2012a)). This effect is also reflected in an increase of the power of the population activity at small frequencies (see appendix figure B.7). These results suggest that on hardware temporal noise is responsible for the increase of correlations and population power in the $\text{FB}_{\text{replay}}$ case (compare $\text{FB}_{\text{replay}}$ to FB in figure 6.5, 6.7 and 6.8).

B.7 Linear model

We investigate the consistency of our results with a linear rate model that allows us to numerically calculate the average correlations from a given connectivity matrix $\mathbf{W}$. The model is defined as (according to, e.g., Grytskyy et al. (2013a))

$$\mathbf{r}(t) = (\mathbf{W}(\mathbf{r} + \mathbf{x}) * h)(t) \quad . \quad (\text{B.3})$$

Here, $\mathbf{r}(t)$ denotes the rate of the individual neurons and $\mathbf{x}(t)$ a Gaussian white noise input that is independent for each neuron. The linear filter kernel $h(t)$ depends on the details of the model, is not relevant in our calculation, and hence is not further specified here. equation (B.3) can be transformed to Fourier domain, where the input and output spectral matrices can be expressed by

$$\mathbf{C}_{\text{RR}}(\omega) = T(\omega)T(\omega)^\dagger \quad , \quad (\text{B.4})$$

$$\mathbf{C}_{\text{RR}}^{\text{in}}(\omega) = \mathbf{W}\mathbf{C}_{\text{RR}}(\omega)\mathbf{W}^T \quad , \quad (\text{B.5})$$

with $T(\omega) = (1 - H(\omega)\mathbf{W})^{-1}$ (Grytskyy et al., 2013a). In the RAND case, the linear equation for the rate of the (unconnected) neurons reads

$$\mathbf{q}(t) = (\mathbf{W}(\tilde{\mathbf{r}} + \mathbf{x}) * h)(t) \quad , \quad (\text{B.6})$$

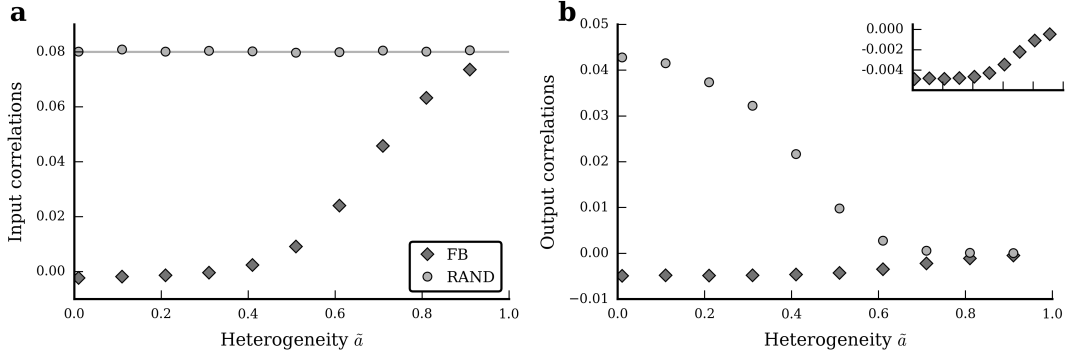


FIGURE B.8: Dependence of population-averaged input correlations **(a)**, and output correlations **(b)** on the absolute magnitude of the effective weights (“heterogeneity” $\tilde{a}$, see text), for the intact network (FB, dark gray diamonds) and the RAND (light gray circles) case. Gray curve in (a) depicts shared-input correlations in a homogeneous network (equation (6.2)). The inset in (b) shows a magnified view of the spike-train correlations in the FB case (dark gray diamonds). Note that in the linear model the $\text{FB}_{\text{replay}}$ is identical to the FB case, and is hence not shown.

where $\tilde{\mathbf{r}}(t)$ has the same auto-correlations as $\mathbf{r}(t)$ but zero cross correlations, i.e., $\mathbf{C}_{\tilde{\mathbf{R}}\tilde{\mathbf{R}}} = \text{diag}(\mathbf{C}_{\mathbf{R}\mathbf{R}})$, since the randomization of spike times removes all spatio-temporal correlations. According to Tetzlaff et al. (2012a) spectral matrices in the RAND case are given by

$$\mathbf{C}_{\mathbf{Q}\mathbf{Q}}^{\text{in}}(\omega) = \mathbf{W}\mathbf{C}_{\tilde{\mathbf{R}}\tilde{\mathbf{R}}}\mathbf{W}^T, \quad (\text{B.7})$$

$$\mathbf{C}_{\mathbf{Q}\mathbf{Q}}(\omega) = |H(\omega)|^2(\mathbf{C}_{\mathbf{Q}\mathbf{Q}}^{\text{in}} + \rho). \quad (\text{B.8})$$

We calculate the population-averaged power- and cross-spectra from the full matrices:

$$\bar{A}_X(\omega) = \frac{1}{N} \sum_i C_{\mathbf{X}\mathbf{X},ii}, \quad (\text{B.9})$$

$$\bar{C}_{\mathbf{X}\mathbf{X}}(\omega) = \frac{1}{N(N-1)} \sum_{i \neq j} C_{\mathbf{X}\mathbf{X},ij}. \quad (\text{B.10})$$

Here, $X \in \{R, Q\}$ denotes the FB and RAND case, respectively. The low frequency coherence is the cross-spectra normalized by the power spectra:

$$\kappa_X(0) = \frac{\bar{C}_{\mathbf{X}\mathbf{X}}(0)}{\bar{A}_X(0)}. \quad (\text{B.11})$$

Note that in the linear model we are actually taking the zero frequency coherence.

As in the spiking model, we consider a sparse network, i.e., we randomly choose for each neuron $i \in [1, N]$ an identical number of presynaptic partners ($K = 15$). In the linear model we do not consider a distribution of non-zero effective weights. Instead, each realized connection is assigned the same weight value $-w$. To mimic the effect of calibration we vary the absolute value of the effective weight by scaling the weights of the non-zero connections with a sigmoidal function of $\tilde{a} \in [0, 1]$:

$$\tilde{w} = \frac{1}{1 + e^{10 \times (\tilde{a} - 0.5)}} w. \quad (\text{B.12})$$

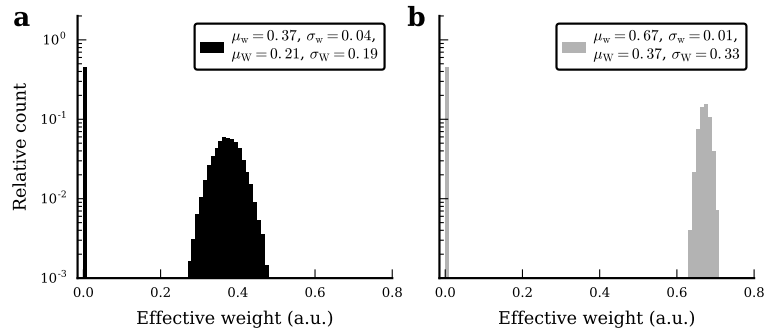


FIGURE B.9: Distributions of effective weights in a sparse network. Different values for the mean μ_w and standard deviation σ_w of the distribution of non-zero weights in (a) and (b), respectively, result in different values for the mean μ_W and standard deviation σ_W of the effective weight matrix.

This procedure changes the variance of the weight matrix (Bernacchia and Wang, 2013) and hence $\tilde{a}$ is denoted the *heterogeneity* of the network. More homogeneous (heterogeneous) networks have larger (smaller) effective weights and hence stronger (weaker) feedback. We obtain qualitatively similar results as we observe on the *Spikey* chip (compare appendix figure B.8 to figure 6.8). Output correlations in the RAND case decrease, while both input and output correlations in the FB case increase with network heterogeneity, i.e., with the variance of the effective weight matrix. Note that, in this simplified model, input correlations in the RAND case are not affected by the level of "heterogeneity" because the non-zero weights are homogeneous, i.e. have zero variance, irrespectively of $\tilde{a}$ (cf. equation (6.3)).

In appendix figure B.9 we illustrate, how in a sparse network the variance of the weight matrix can increase, although the distribution of non-zero weights becomes narrower. The standard deviation σ_w of a distribution of non-zero weights with mean μ_w is (in this example) smaller than the standard deviation σ_W of the full effective weight matrix, due to the sparseness of the matrix (appendix figure B.9a; here, we chose $\epsilon = 0.8$). If we, at the same time, increase the mean μ_w and decrease the standard deviation σ_w of non-zero weights, the standard deviation of the weight matrix σ_W can increase significantly (appendix figure B.9). While the distribution of effective weights is broadened, the mean is decreased, which has a greater impact on the size of correlations. This observation could explain the decrease of correlations with increased calibration of the neuromorphic chip.

B.8 Firing statistics in the RAND case

The firing rate distributions in the RAND case are similar to those in the FB scenario (compare appendix figure B.10 to figure 6.6d–f). As in the FB case they become broader for increasing heterogeneity. Due to a larger variance of the membrane potentials, inactive neurons in the FB case have a higher probability to fire in the RAND case (compare gray bars in horizontal histograms of appendix figure B.10a–c and figure 6.6d–f). Neurons with firing rates larger than the average firing rate are strongly driven by constant current influx, and hence show similar firing rates than in the FB scenario. The irregularity of firing increases for the RAND compared to the FB case. This can also be traced back to an increased variance of the membrane potential for the RAND case (data not shown).

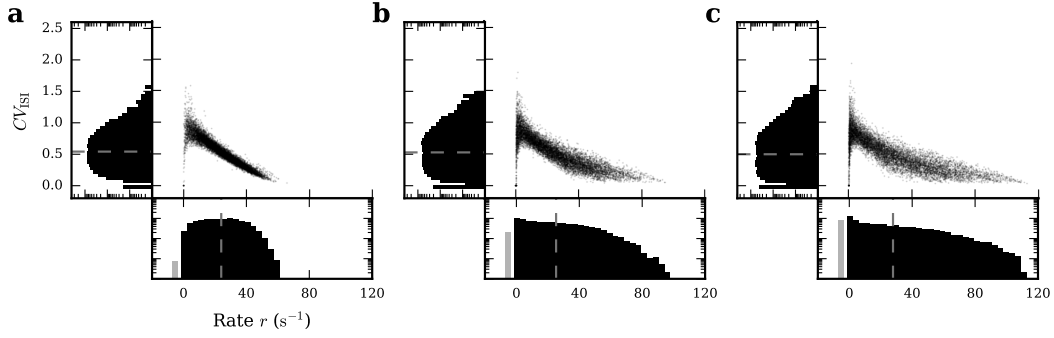


FIGURE B.10: **(a–c)** Like figure 6.6d–f, but for the RAND case. The mean of firing rate (24.0 s^{-1} , 25.2 s^{-1} , 27.8 s^{-1}) and CV_{ISI} distributions (0.54, 0.53, 0.5) are marked with dashed lines. Percentage of dead neurons: 0.00%, 0.02%, 0.08%.

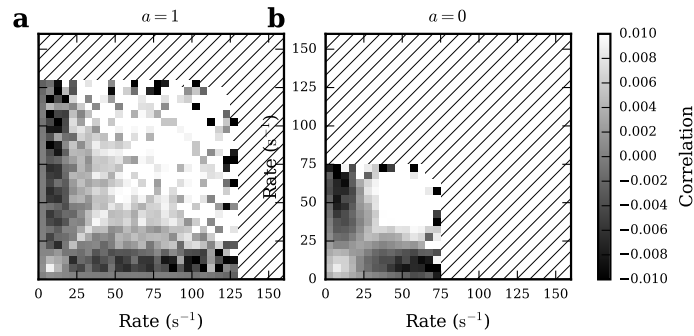


FIGURE B.11: Firing-rate dependence of pairwise low-frequency spike-train coherence in the uncalibrated **(a)** and fully calibrated system **(b)**. Each bin represents the low-frequency spike-train coherence $\kappa_{ij} = \frac{1}{f_{\max} - f_{\min}} \int_{f_{\min}}^{f_{\max}} \frac{\Re(C_{ij}(f))}{\sqrt{A_i(f)A_j(f)}} df$, averaged across all neuron pairs $\{i, j\}$ with firing rates in the interval $[r_i, r_i + \Delta r)$ and $[r_j, r_j + \Delta r)$. Here, $\Re(C_{ij}(f))$ denotes the real part of the complex cross-spectrum at frequency f . $f_{\min} = 0.1 \text{ Hz}$, $f_{\max} = 20 \text{ Hz}$, $\Delta r = 5 \text{ s}^{-1}$. Data averaged across $M = 100$ network realizations.

B.9 Correlation matrices

In addition to the population-averaged measures from the main manuscript, we calculated the pairwise correlations for each pair i, j of neurons with $i \neq j \in [1, N]$, and ordered these by the time-averaged rate of the corresponding neurons (appendix figure B.11). This reveals a dependence of the pairwise correlation on the firing rate of the respective neurons. If both neurons fire at low rate (here $< 5 \text{ s}^{-1}$), correlations will be close to zero similar to the results in De la Rocha et al. (2007). For high rates (here $> 25 \text{ s}^{-1}$) we find mostly positive correlations. However, for neurons firing at intermediate rates, the activity of the pair is often anti-correlated, and can hence effectively suppress (positive) shared-input correlations. After calibration, the amount of neurons firing at intermediate rates and showing negative correlations increases (appendix figure B.11b, figure 6.6). Shared-input correlations are hence suppressed by more neurons leading to smaller input correlations than in the uncalibrated case (figure 6.8).

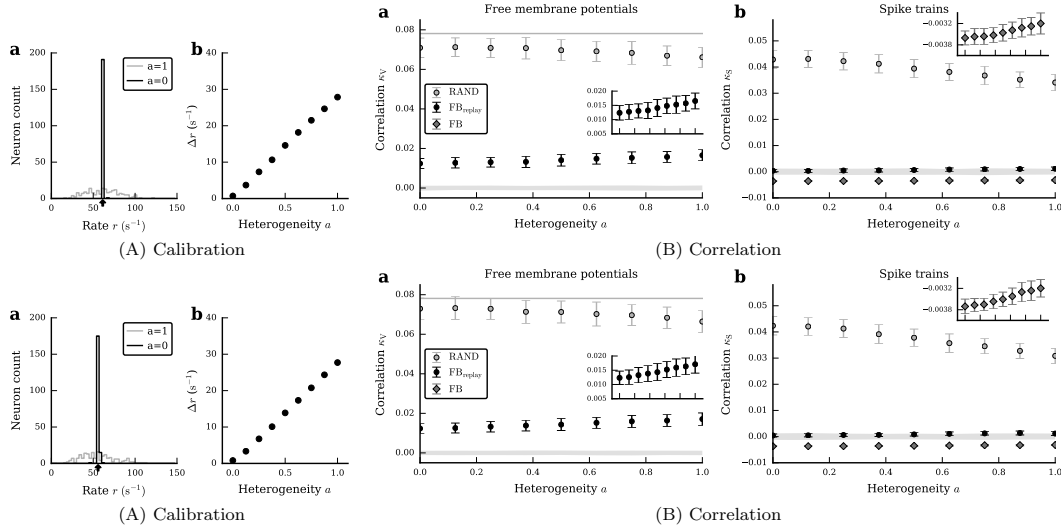


FIGURE B.12: Like figure 6.4 and 6.8 in (A) and (B), respectively, but for two additional chips of version 5.

B.10 Results for different *Spikey* chips

We have performed the same experiments as described in the main text using two additional *Spikey* chips of the current version (5). Different chips show different realizations of fixed-pattern noise, and hence calibration was repeated for each chip separately (figure 6.4, appendix figure B.12A). For all chips we find qualitatively the same results as described above: in the FB case, free membrane potentials (and spike trains) are decorrelated by inhibitory feedback and correlations increase with an increasing level of heterogeneity (figure 6.8, appendix figure B.12B).

This also applies to the results of identical experiments on three previous-generation chips (version 4). These old chips show more pronounced heterogeneity in their parameters. In addition, we observe larger quantitative differences between old chips for the uncalibrated case, which is likely to be caused by different extents of intrinsic fixed-pattern noise. Unfortunately, the old systems show an artifact affecting the population power at around 100 Hz (appendix figure B.13), consistently across all chips of identical revision. We can exclude that this effect can be traced back to network effects (it occurs across the FB, FB_{replay} and RAND cases) or to single-cell power spectra (data not shown). We hence have to consider it an artificial spurious synchronization. As the artifact lies outside the range of frequencies that are relevant for our measures of correlation it does not affect the main results and experiments on the old chips qualitatively confirm our results in the main manuscript (compare appendix figure B.14 to figure 6.4, 6.8 and appendix figure B.12).

B.11 Reproducibility of networks with intact feedback

We measured the standard deviation of free membrane potential and spike train correlations over several trials for a single network realization (appendix figure B.15). This standard deviation is smaller than the standard deviation we observe over different network realizations (compare to figure 6.8 and appendix figure B.12), which indicates that the latter is mostly caused by the different realizations of the connectivity, not by trial-to-trial variability. Note that the variability between trials of

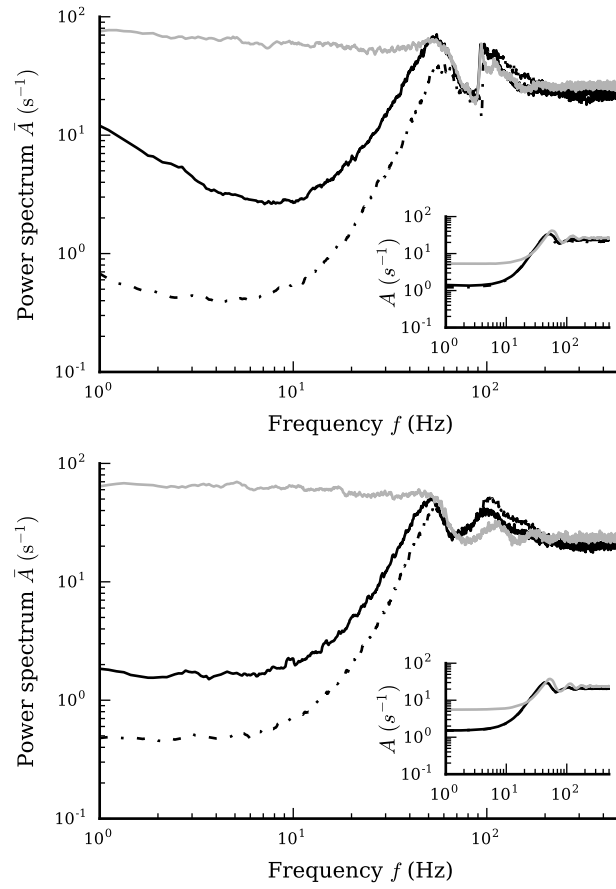


FIGURE B.13: Spectra of trial-averaged population power $\bar{A}(f)$ and trial-averaged single-cell power $A(f)$ for chip 666 (version 4, left) and chip 508 (version 5, right) illustrating the synchronization artifact in the 100 Hz range for the intact network (FB, dark gray), the FB_{replay} (black) and the RAND (light gray) case.

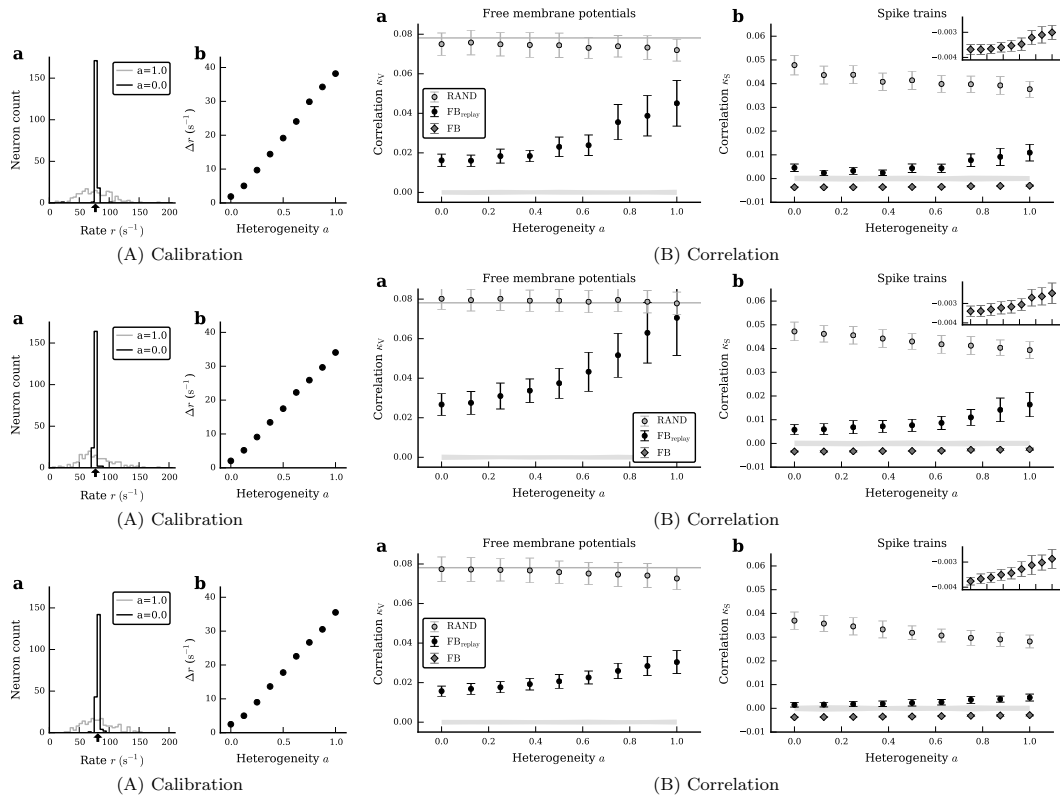


FIGURE B.14: Like figure 6.4 and 6.8 in (A) and (B), respectively, but for three chips of version 4.

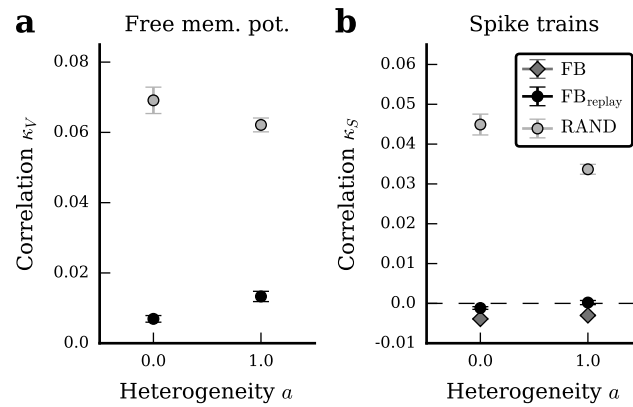


FIGURE B.15: Like figure 6.8 and for the same chip, but for $L = 20$ trials of one network realization.

networks with intact feedback is likely to be larger than between replays of network activity as shown in figure 6.3, because network dynamics may be chaotic. The data shown in appendix figure B.15 has to be interpreted with care, because the reproducibility of only a single network realization is considered. For different realizations, the mean and standard deviation will in general be different.

B.12 Extraction of effective PSP amplitudes, time constants and delays

In order to provide an estimation for the order of magnitude of various neuron parameters, we fitted the postsynaptic potentials (PSPs) of current-based LIF neurons with exponential postsynaptic currents (PSCs) to the spike-triggered average (STA) of free membrane potentials obtained from hardware emulations. The differential equations describing the dynamics of membrane potentials $V_j(t)$ and synaptic currents $I_j(t)$ for this simplified model are given by

$$\tau_m^{\text{eff}} \dot{V}_j(t) = -V_j(t) + RI_j(t) \quad (\text{B.13})$$

$$\tau_{\text{syn}}^{\text{eff}} \dot{I}_j(t) = -I_j(t) + \tau_{\text{syn}}^{\text{eff}} \sum_k J_{jk} s_k(t - d^{\text{eff}}). \quad (\text{B.14})$$

where R is the resistance of the membrane, τ_m^{eff} the membrane time constant, $\tau_{\text{syn}}^{\text{eff}}$ the synaptic time constant and J_{jk} the synaptic weight of the connection from neuron k to j . Here the superscript eff indicates that these are not the parameters realized on hardware, but of an abstract model that we fit to hardware measurements. Data were obtained from 100 network realizations of the RAND case (same dataset as for figure 6.8). For each neuron we computed the STA of its free membrane potential individually for each spike train of its presynaptic partners. The PSP for the current based model described above is given by

$$V(t) = V_{\text{max}}^{\text{eff}} \frac{\tau_{\text{syn}}^{\text{eff}}}{\tau_{\text{syn}}^{\text{eff}} - \tau_m^{\text{eff}}} \left(e^{-(t-d^{\text{eff}})/\tau_{\text{syn}}^{\text{eff}}} - e^{-(t-d^{\text{eff}})/\tau_m^{\text{eff}}} \right) \Theta(t - d^{\text{eff}}), \quad (\text{B.15})$$

where we introduced $V_{\text{max}}^{\text{eff}} = RJ$ to simplify the fitting procedure. For each pair of neurons i, j we obtain a set of parameters: $V_{\text{max}}^{\text{eff}}$, $\tau_{\text{syn}}^{\text{eff}}$, τ_m^{eff} and d^{eff} .

The fitted curves (using the function `scipy.optimize.curve_fit` with default parameters from Python's `scipy` module v0.17.0) match fairly well to the empirical data to allow an estimation of the *effective* values of the neuron parameters governing the shape of the postsynaptic potentials (appendix figure B.16a). However, note that the resulting parameter distributions do not precisely reflect the actual parameter distributions on the chip.

The distributions of neuron and synapse parameters change slightly with calibration. For the calibrated network the absolute values of $V_{\text{max}}^{\text{eff}}$, $\tau_{\text{syn}}^{\text{eff}}$, τ_m^{eff} and d^{eff} are larger than for the uncalibrated network (see table B.2 and appendix figure B.16b).

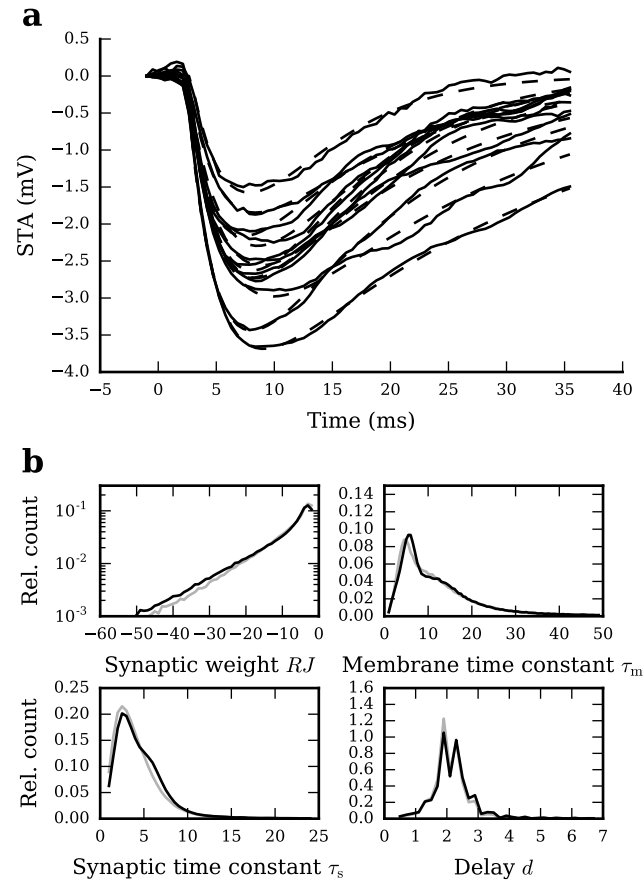


FIGURE B.16: **(a)** Fits of theoretical PSPs (dashed; for model, see equation (B.15)) to STAs of free membrane potentials in the FB case (solid). **(b)** Distributions of effective parameters obtained from fitting. Black and gray lines correspond to the fully calibrated ($a = 0$) and uncalibrated ($a = 1$) network, respectively. Median, 25th and 75th percentile of all fitted parameters are reported in table B.2.

A		Model summary	
Populations		One (inhibitory)	
Topology		—	
Connectivity		Random convergent connections (fixed in-degree)	
Neuron model		Leaky integrate-and-fire (LIF), fixed firing threshold, fixed absolute refractory time	
Channel models		—	
Synapse model		Exponentially decaying currents, fixed delays	
Plasticity		—	
External input		Resting potential higher than threshold (= constant current) ($E_l > \Theta$)	
Measurements		Spikes and membrane potentials	
Other		No autapses, no multapses	
B		Populations	
Name		Elements	Size
I		LIF neuron	N
C		Connectivity	
Source	Target	Pattern	
I	I	Random convergent connect, in-degree K	
D		Neuron and synapse model	
Type		Leaky integrate-and-fire, exponential currents	
Subthreshold dynamics		Subthreshold dynamics ($t \notin (t^*, t^* + \tau_{\text{ref}})$): $C_m \frac{\text{d}}{\text{d}t} v(t) = -g_l(v(t) - E_l) + I_{\text{syn}}(t)$ Reset and refractoriness ($t \in (t^*, t^* + \tau_{\text{ref}})$): $v(t) = v_{\text{reset}}$	
Current dynamics		$\tau_{\text{syn}} \frac{\text{d}}{\text{d}t} I_{\text{syn}}(t) = -I_{\text{syn}}(t) + \sum_{i,k} J \delta(t - t_i^k)$ Here the sum over i runs over all presynaptic neurons and the sum over k over all spike times of the respective neuron i	
Spiking		If $v(t^* -) < \Theta \wedge v(t^* +) \geq \Theta$: emit spike with time stamp t^*	
E		Measurements	
Spike trains		recorded from P_s neurons	
Membrane potentials		recorded from P_v neurons	

TABLE B.5: Description of the network model (according to Nordlie, Gewaltig, and Plesser (2009b)).

B Populations		
Name	Values	Description
N	$\{192, 4800\}$	network size
C Connectivity		
Name	Values	Description
K	$\{15, 375\}$	number of presynaptic partners
D Neuron		
Name	Values	Description
C_m	0.2 nF	membrane capacitance
τ_{ref}	0.1 ms	refractory period
v_{reset}	-80 mV	reset potential
E_l	-52 mV	resting potential
Θ	$\sim \mathcal{N}(\mu, \sigma^2)$, $\mu = -62\text{mV}$, $\sigma \in [0, 8.8]\text{mV}$	firing threshold
g_l	10 nS	leak conductance
D Synapse		
Name	Values	Description
τ_{syn}	5 ms	synaptic time constant
J	-0.254 nA	synaptic weight
d	1.0 ms	synaptic delay
E Measurements		
Name	Values	Description
P_s	$\{192, 4800\}$	number of neurons spike trains are recorded from
P_v	$\{150, 150\}$	number of neurons membrane potentials are recorded from

TABLE B.6: Parameter values for the network model described in appendix table B.5 with distributed thresholds.

B Populations		
Name	Values	Description
N	192	network size
C Connectivity		
Name	Values	Description
K	15	number of presynaptic partners
D Neuron		
Name	Values	Description
C_m	0.2 nF	membrane capacitance
τ_{ref}	0.1 ms	refractory period
v_{reset}	−80 mV	reset potential
E_l	−52 mV	resting potential
Θ	−62 mV	firing threshold
g_l	10 nS	leak conductance
D Synapse		
Name	Values	Description
τ_{syn}	5 ms	synaptic time constant
J	$\sim [\mathcal{N}(0.254, [0.00254, 0.75])]_+$	synaptic weight, clipped to positive values
d	1.0 ms	synaptic delay
E Measurements		
Name	Values	Description
P_s	192	number of neurons spike trains are recorded from
P_v	150	number of neurons membrane potentials are recorded from

TABLE B.7: Parameter values for the network model described in appendix table B.5 with distributed weights.

A		Model summary	
Populations		Two (excitatory, inhibitory)	
Topology		—	
Connectivity		Random convergent connections (fixed in-degree)	
Neuron model		Leaky integrate-and-fire (LIF), fixed firing threshold, fixed absolute refractory time	
Channel models		—	
Synapse model		Exponentially decaying currents, fixed delays	
Plasticity		—	
External input		Resting potential higher than threshold (= constant current) ($E_l > \Theta$)	
Measurements		Spikes and membrane potentials	
Other		No autapses, no multapses	
B		Populations	
Name	Elements	Size	
E	LIF neuron	N_E	
I	LIF neuron	N_I	
C		Connectivity	
Source	Target	Pattern	
E	E	Random convergent connect, in-degree K_E , weight J_E	
E	I	Random convergent connect, in-degree K_E , weight J_E	
I	E	Random convergent connect, in-degree K_I , weight J_I	
I	I	Random convergent connect, in-degree K_I , weight J_I	
D		Neuron and synapse model	
Type	Leaky integrate-and-fire, exponential currents		
Subthreshold dynamics	Subthreshold dynamics ($t \notin (t^*, t^* + \tau_{\text{ref}})$): $C_m \frac{\text{d}}{\text{d}t} v(t) = -g_l(v(t) - E_l) + I_{\text{syn}}(t)$ Reset and refractoriness ($t \in (t^*, t^* + \tau_{\text{ref}})$): $v(t) = v_{\text{reset}}$		
Current dynamics	$\tau_{\text{syn}} \frac{\text{d}}{\text{d}t} I_{\text{syn}}(t) = -I_{\text{syn}}(t) + \sum_{i,k} J \delta(t - t_i^k)$ Here the sum over i runs over all presynaptic neurons and the sum over k over all spike times of the respective neuron i		
Spiking	If $v(t^* -) < \Theta \wedge v(t^* +) \geq \Theta$: emit spike with time stamp t^*		
E		Measurements	
Spike trains		recorded from P_s^E excitatory and P_s^I inhibitory neurons	
Membrane potentials		recorded from P_v^E excitatory and P_v^I inhibitory neurons	

TABLE B.8: Description of the network model consisting of an excitatory and an inhibitory population (according to Nordlie, Gewaltig, and Plesser (2009b)).

B Populations		
Name	Values	Description
N_E	96	size of the excitatory population
N_I	96	size of the inhibitory population
C Connectivity		
Name	Values	Description
K_E	7	number of excitatory presynaptic partners
K_I	8	number of inhibitory presynaptic partners
D Neuron		
Name	Values	Description
C_m	0.2 nF	membrane capacitance
τ_{ref}	0.1 ms	refractory period
v_{reset}	−80 mV	reset potential
E_l	−52 mV	resting potential
Θ	−62 mV	firing threshold
g_l	10 nS	leak conductance
D Synapse		
Name	Values	Description
τ_{syn}	5 ms	synaptic time constant
J_E	0.0635 nA	excitatory synaptic weight
J_I	−0.254 nA	inhibitory synaptic weight
d	1.0 ms	synaptic delay
E Measurements		
Name	Values	Description
P_s^E	96	number of excitatory neurons spike trains are recorded from
P_s^I	96	number of inhibitory neurons spike trains are recorded from
P_v^E	150	number of excitatory neurons membrane potentials are recorded from
P_v^I	150	number of inhibitory neurons membrane potentials are recorded from

TABLE B.9: Parameter values for the network model described in appendix table B.8.

Appendix C

Supplementary material to “Extremely scalable spiking neural network simulation code: from laptops to exascale computers”

C.1 Network and simulation parameters

C.2 Connection infrastructure for devices

In order to avoid extensive communication between nodes, stimulation and recording devices should be co-located on the same compute node as the neurons to which they are sending events and from which they are receiving events, respectively. In NEST, creation of a single stimulation or recording device leads to a creation of a device object on every thread. These device objects only send data to and receive data from neurons located on the same thread. This has two advantages: On the one hand, processes do not need to exchange data that is sent from and received by devices, and on the other hand this supports parallel post-processing of recorded data, since the data is already distributed over compute nodes. To simplify maintenance and construction of the two-tier connection infrastructure managing connections between neurons, connections to and from devices are stored in separate data structures that consist of two four-dimensional vectors (figure C.1). Connections from devices are stored according to (i) the thread of the device, (ii) the thread-local id of the device, and (iii) the synapse type. If a device sends an event (e.g., a spike generated by a Poisson generator), it is delivered through all connections for all synapse types to the corresponding targets. Connections to devices are stored according to (i) the thread of the sending neuron, (ii) the thread-local id of the neuron and (iii) the synapse type. If a neuron sends an event, it is delivered through all connections for all synapse types to the corresponding target devices. Due to their small memory footprint, these data structures are not included in the memory model.

The separate data structures for connections to and from devices can improve the delivery of data from devices to neurons, for example from Poisson sources, with regard to the previous kernel. While the previous technology uses the same infrastructure employed by neurons to communicate spikes, devices have now a more direct access to their targets.

A: Model Summary		
Populations	Two: excitatory, inhibitory	
Topology	—	
Connectivity	Random convergent connections	
Neuron models	Leaky integrate-and-fire (LIF), fixed firing threshold, fixed absolute refractory time (voltage clamp), α -currents	
Channel models	—	
Synaptic plasticity	Spike-timing dependent plasticity at excitatory-excitatory recurrent connections	
External input	Independent fixed-rate Poisson spike trains to all neurons	
Measurements	Total number of spikes generated per process	
Other	Multapses allowed, no autapses	

B: Populations		
Name	Elements	Size
E	LIF neuron	$N_E = 4N_I$ with $N_E + N_I = N$
I	LIF neuron	N_I

C: Connectivity			
Name	Source	Target	Pattern
EE	E	E	Random convergent, $K_E \rightarrow 1$, weight variable, delay d
IE	E	I	Random convergent, $K_E \rightarrow 1$, weight J , delay d
EI	I	E	Random convergent, $K_I \rightarrow 1$, weight $-gJ$, delay d
II	I	I	Random convergent, $K_I \rightarrow 1$, weight $-gJ$, delay d

TABLE C.1: Tabular description of network model after Nordlie, Gewaltig, and Plesser (2009b).

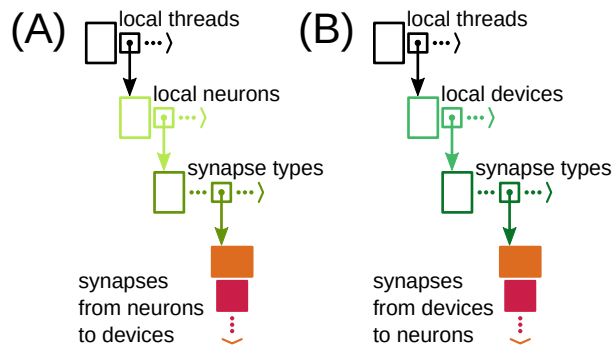


FIGURE C.1: **Connection infrastructure for devices of the NEST 5g kernel.** (A) Connections from neurons to thread-local devices. Each MPI process stores a pointer to a dynamic container of connection objects for every thread, every thread-local neuron, and every synapse type. If a synapse type is in use, the corresponding container (orange filled rectangle) holds the synapses objects (pink filled squares). (B) Connections from devices to thread-local neurons. A data structure of identical shape is used to store the connections from devices to thread-local neurons.

D: Neuron Model		
Name	LIF neuron	
Type	Leaky integrate-and fire, α -current input	
Subthreshold dynamics	$\tau_m \frac{dV(t)}{dt} = -V(t) + \frac{I(t)}{C_m}$ if $(t > t^* + \tau_{ref})$ $V(t) = V_{reset}$ else	
Synaptic current dynamics	For each presynaptic spike: $I_{syn}(t) = w \frac{e}{\tau_{syn}} t e^{-t/\tau_{syn}}$	
Spiking	If $V(t-) < \Theta$ and $V(t+) \geq \Theta$ 1. set $t^* = t$ 2. emit spike with time stamp t^*	
E: Synapse Model		
Name	Power law STDP (Morrison, Aertsen, and Diesmann, 2007a)	
Type	Weight dependent STDP with a power law update rule for potentiation and a multiplicative update rule for depression	
Spike pairing scheme	All-to-all (for nomenclature see Morrison, Diesmann, and Gerstner, 2008)	
Pair-based update rule	$\Delta w^+ = F_+(w) e^{-\frac{ \Delta t }{\tau_+}}$ if $\Delta t > 0$ $\Delta w^- = -F_-(w) e^{-\frac{ \Delta t }{\tau_-}}$ else Δt - temporal difference between post- and pre-synaptic spikes, synaptic delay considered to be 50% axonal	
Weight dependence	$F_+(w) = \lambda w_0^{1-\mu} w^\mu, \quad F_-(w) = \lambda \alpha w$	
F: Input		
Type	Target	Description
Poisson generator	I&E	Independent for each neuron, rate $\nu_x K_x$, weight J

TABLE C.2: Tabular description of network model after Nordlie, Gewaltig, and Plesser (2009b), continued.

B: Populations		
Name	Value	Description
N_E	variable	Size of excitatory population E
N_I	variable	Size of inhibitory population I
C: Connectivity		
Name	Value	Description
K_E	9000	Number of incoming connections from E
K_I	2250	Number of incoming connections from I
D: Neuron Model		
Name	Value	Description
τ_m	10 ms	Membrane time constant
C_m	250 pF	Membrane capacitance
Θ	20 mV	Fixed firing threshold
V_0	0 mV	Resting potential
V_{reset}	V_0	Reset potential
τ_{ref}	0.5 ms	Absolute refractory period
τ_{syn}	0.33 ms	Rise time of PSC
μ_V	5.7 mV	Mean value of initial normal distribution of membrane potentials
σ_V	7.2 mV	Standard deviation of initial normal distribution of membrane potentials
E: Synapse Model		
Name	Value	Description
τ_+	15 ms	Time constant of potentiation window
τ_-	30 ms	Time constant of depression window
λ	0.1	Learning rate
μ	0.4	Weight-dependence parameter of potentiation
α	0.0513	Asymmetry parameter
w_0	1 pA	Normalization parameter
μ_w	38.58 pA	Mean value of initial normal distribution of synaptic weights
σ_w	3.47 pA	Standard deviation of initial normal distribution of synaptic weights
d	1.5 ms	Delay
J	0.14 mV	Excitatory weight
g	5.0	Relative inhibitory weight
F: Input		
Name	Value	Description
K_x	9000	Number of external inputs
ν_x	$2.3 \frac{\text{spikes}}{\text{s}}$	Rate of single external input

TABLE C.3: Simulation parameters after Nordlie, Gewaltig, and Plesser (2009b).

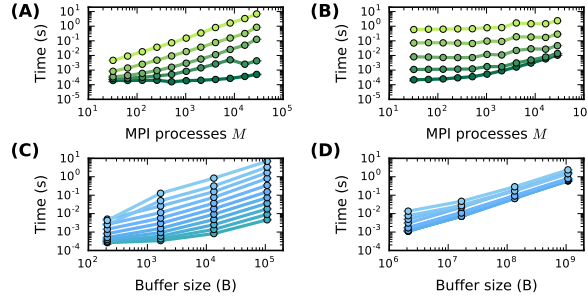


FIGURE C.2: **Scaling of communication time for *MPI_Allgather* (left) and *MPI_Alltoall* (right) over MPI processes (top) and send buffer sizes (bottom).** Darker colors indicate smaller buffer sizes (top) and smaller number of MPI processes (bottom), respectively. Number of MPI processes $M \in \{32; 64; 128; 256; 512; 1024; 2048; 4096; 8192; 16,384; 28,672\}$. Send buffer sizes for *MPI_Allgather* $s \in \{0\text{kB}; 0.2\text{kB}; 1.6\text{kB}; 13\text{kB}; 105\text{kB}\}$, for *MPI_Alltoall* $s \in \{0\text{kB}; 2048\text{kB}; 16,384\text{kB}; 131,072\text{kB}; 1,048,576\text{kB}\}$. Error bars denote SEM and are partly covered by markers.

C.3 Scalability of *MPI_Allgather* and *MPI_Alltoall*

Here we investigate the scaling properties of *MPI_Allgather* and *MPI_Alltoall* on JUQUEEN for increasing numbers of MPI processes and MPI send buffer sizes. In network simulators, naively switching from a source-based AER using *MPI_Allgather* to a target-based AER scheme using *MPI_Alltoall* would lead to an average increase of the send buffer size proportional to the average out-degree of neurons independent of the number of processes, as in the case of 5g-nosort. Here we assume that the out-degree is $K = 10,000$ and hence increase the send buffer size for all *MPI_Alltoall* measurements correspondingly. We measure the average time a single MPI communication among M nodes takes for an MPI send buffer of size s . In addition we measure the latency by setting the send buffer size to zero.

For *MPI_Allgather*, the latency is almost constant, independent of the number of processes. For non-empty send buffers however, the wall-clock time of a single MPI call increases significantly with the number of processes, almost linearly for large send buffer sizes (figure C.2). Even though the send buffers are very small, the size of receive buffers scales linearly in the number of MPI processes. The increase in communication time for *MPI_Allgather* is most likely due to the increase in receive buffer sizes that require an increasing amount of memory to be written. For *MPI_Alltoall*, on the other hand, latency increases significantly with the number of processes while scaling of communication time for non-empty buffers is much better than for *MPI_Allgather*, with only a small increase over the range of investigated processes. Note that the total size of the send buffers is much larger for *MPI_Alltoall* with a maximal send buffer size of 1GB that is communicated across 28,672 compute nodes in $\sim 2\text{s}$. From 10^4 processes on, the receive buffers of *MPI_Allgather* are larger than for *MPI_Alltoall*, leading to a better performance of *MPI_Alltoall* (figure C.2A,B). When scaling the buffer size, communication time increases supra-linearly for *MPI_Allgather*, while it increases almost linearly for *MPI_Alltoall*, almost independent of the number of MPI processes (figure C.2C,D).

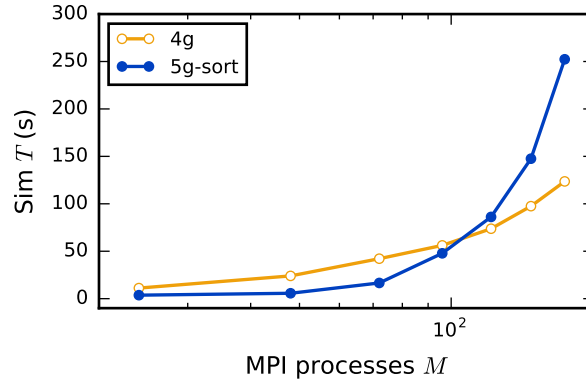


FIGURE C.3: **Weak scaling of a neuronal network simulation with structural plasticity on JURECA.** Runtime for simulating 5000 neurons per compute node for 10s of biological real time using one thread per MPI process. $M \in \{24; 48; 72; 96; 120; 144; 168\}$. Color code same as in figure 8.7.

C.4 Structural plasticity

Structural plasticity modifies the connectivity of the network at runtime, for example allowing researchers to grow network connectivity according to certain rules instead of specifying it at the beginning of the simulation (Diaz-Pier et al., 2016). Due to the purely postsynaptic storage of connections in the previous simulation kernel, structural plasticity only needed to manipulate postsynaptic data structures. However, due to the two-tier connection infrastructure in the new kernel, it becomes necessary to update also the presynaptic part when connections are created or removed at runtime. Since the scalability of the structural plasticity algorithm in the previous kernel is only feasible up to a few hundreds of processes, we provide only a naive implementation for the new simulation kernel. In this implementation, the whole presynaptic part of the connection infrastructure is destroyed whenever new connections are created or connections are removed and then reconstructed from scratch from the postsynaptic part. While this is a fairly expensive operation, structural plasticity operates on slow timescales in the seconds range, making this approach feasible up to about $\mathcal{O}(100)$ MPI processes. In this range the performance is comparable to the previous simulation kernel figure C.3. To support larger simulations, an alternative implementation needs to be developed, that in the optimal case operates only on local data structures, without requiring global communication among all compute nodes.

C.5 Strong scaling over threads

Modern simulation codes need to be able to exploit the increased parallelism available on compute nodes that support tens to hundreds of concurrent threads. Here we investigate strong scaling over threads for the previous and the new kernel. For an increasing number of threads the build time scales very well for both kernel versions, as long as the number of threads is smaller or equal to the number of available cores (figure C.4). As soon as hyper-threading is used, the scaling behavior becomes slightly worse, leading to smaller differences between 32 and 64 threads. As nodes are constructed serially, this contribution does not scale over threads but is in any case negligible compared to the time spent on the construction of connections. The presim time does not scale with the number of threads for the new kernel and even

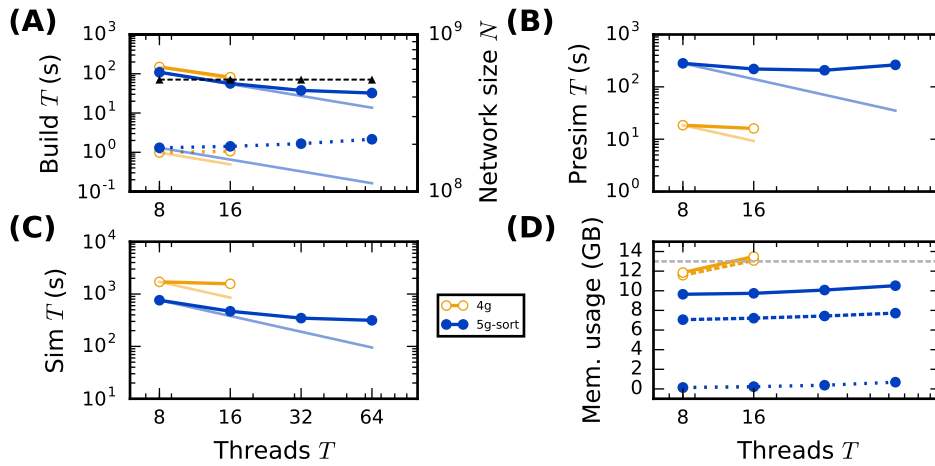


FIGURE C.4: **Strong scaling over threads of an archetypal neuronal network simulation on a petascale computer.** Runtime and memory usage per compute node for an increasing number of threads per MPI process ($T \in \{8; 16; 32; 64\}$) and $M = 28672$ MPI processes on JUQUEEN. Same line styles and color code as in figure 8.7. Other parameters same as in figure 8.7. Solid lines indicate decrease of the respective times for perfect scaling.

increases for a large number of threads that exceeds the core count. As in the weak scaling scenario, most of the time is spent on collocating MPI buffers which does not significantly profit from multiple threads and is mainly limited by memory access not processing speed. The sim time decreases monotonously for the new kernel, even beyond 16 threads in contrast to the old kernel. For both kernels, the memory consumption increases with the number of threads, not allowing simulations with more than 16 threads when employing the old kernel. Further detailed investigations are necessary to improve scaling of the presim time with an increasing number of threads in the new kernel.

C.6 Strong scaling over MPI processes (full data)

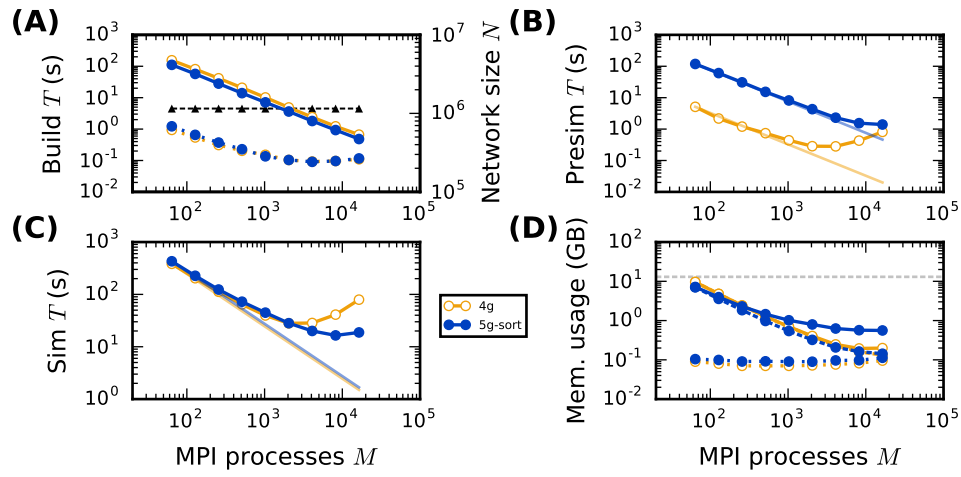


FIGURE C.5: **Strong scaling of an archetypal neuronal network simulation on a petascale computer.** Same lines styles and color codes as in figure 8.9.

Appendix D

Supplementary material to “Interfacing neural network simulators and the OpenAI Gym”

D.1 Network description

The tables D.1, D.2, D.3 summarize the network architecture and parameters.

D.2 JSON Message types

Listing D.1 show the standard message types used for communication between the OpenAI Gym and the RMT. All messages are serialized using JSON and communicated via ZeroMQ.

LISTING D.1: Message types used for communication.

```
BasicMsg
{
    float min
    float max
    float value
    float timestamp
}

ObservationMsg
{
    BasicMsg[] observations    # one basic msg per dimension
}

RewardMsg
{
    BasicMsg[] reward         # reward is always one dimensional
}

ActionMsg
{
    BasicMsg[] actions        # one dimensional for discrete actions
                              # or one dimension per possible action
}
```

D.3 Example wrapper configuration file

Listing D.2 shows an example configuration file for running the mountain car environment.

LISTING D.2: Example configuration file for the wrapper to run the “MountainCar-v0” environment.

A		Model summary	
Populations	Seven		
Topology	—		
Connectivity	Population specific		
Neuron model	Linear & threshold-linear rate		
Channel models	—		
Synapse model	Instantaneous & delayed continuous coupling		
Plasticity	Three-factor Hebbian		
External input	Continuous MUSIC ports		
External output	Continuous MUSIC ports		
Measurements	Rates of all neurons		
B			
Populations			
Name	Elements	Size	
Observation	MUSIC in port	1	
Reward	MUSIC in port	1	
Action	MUSIC out port	1	
Place cells	Threshold-linear	16(25)	
Critic	Threshold-linear	1	
Actor	Threshold-linear	4(3)	
Prediction error	Linear	1	
C			
Connectivity			
Source	Target	Pattern	
Observation	Place cells	One-to-one (by MUSIC channel), instantaneous, static, weight w_o	
Reward	Prediction error	One-to-one (by MUSIC channel), instantaneous, static, weight w_r	
Actor	Action	One-to-one (by MUSIC channel), instantaneous, static, weight w_a	
Place cells	Critic	All-to-all, instantaneous, plastic, initial weight w_{pc}	
Place cells	Actor	All-to-all, instantaneous, plastic, initial weight w_{pa}	
Critic	Prediction error	One-to-one, instantaneous, static, weight $1/d - 1/\tau_r$	
Critic	Prediction error	One-to-one, delay d , static, weight $-1/d$	
Actor	Actor	All-to-all, instantaneous, static, weight $\alpha \exp(-\Delta a/\sigma) + \beta$	
D			
Neuron and synapse model			
Type	Linear rate neuron		
Dynamics	$\tau \frac{dz(t)}{dt} = -z(t) + \mu + (h(t) - \theta) + \xi(t)$		
Type	Threshold-linear rate neuron		
Dynamics	$\tau \frac{dz(t)}{dt} = -z(t) + \mu + \Theta(h(t) - \theta)(h(t) - \theta) + \xi(t)$		
Type	Three-factor Hebbian synapse		
Plasticity	$\Delta w_{ij} = \eta \delta(t) g x_j(t - \delta t) \Theta(z_i(t - \delta t) - \theta_{\text{post}})$		
E			
Input			
Type	Description		
Observation	Rate $r \in [-1, 1]$ according to tuning of place cell (using <i>discretize</i> adapter)		
Reward	Rate $r \in [-1, 1]$ according to reward provided by the environment		
F			
Output			
Type	Description		
Action	Rates $r_i \in [0, \infty)$ according to activities of the actor units		

TABLE D.1: Description of the network model (according to (Nordlie, Gewaltig, and Plesser, 2009b)).

B Populations: place cells	
Name	Values
τ	5.0 (1.0)
g	1.0
μ	0.0
σ_ξ	0.0
θ	-0.5
B Populations: critic	
Name	Values
τ	0.1
g	1.0
μ	-1.0
σ_ξ	0.0
θ	-1.0
B Populations: reward	
Name	Values
τ	1.0
g	1.0
μ	0.0
σ_ξ	0.0
θ	0.001 (-0.0999)
B Populations: actor	
Name	Values
τ	0.1
g	1.0
μ	0.0
σ_ξ	0.2 (0.05)
θ	0.0

TABLE D.2: Table of the network parameters used for both tasks (according to (Nordlie, Gewaltig, and Plesser, 2009b)). Values in brackets are used for the *MountainCar* environment.

C Connectivity	
Name	Values
w_o	0.5
w_r	0.1
w_a	1.0
w_{pc}	0.0
$w_{pc}^{\min}$	-1.0
$w_{pc}^{\max}$	1.0
θ_{post}^{pc}	-1.0
w_{pa}	0.9(0.3)
$w_{pa}^{\min}$	0.1(0.05)
$w_{pa}^{\max}$	1.0
θ_{post}^{pa}	0.5(0.1)
d	1.0
τ_r	20000.0
α	1.2
β	-0.55
σ_ξ	0.1
η_{critic}	0.01(0.125)
η_{actor}	0.2(0.250)
δt	19.0(0.0)
E Input: discretize adapter	
Name	Values
σ_x	0.01 (0.2)
σ_y	- (0.2)

TABLE D.3: Table of the network parameters used for both tasks (according to (Nordlie, Gewaltig, and Plesser, 2009b)), continued. Values in brackets are used for the *Mountain-Car* environment.

```
"All":
{
  "seed": 12345,
  "time_stamp_tolerance": 0.01,
  "prefix": null,
  "write_report": true,
  "report_file": "./report.json",
  "overwrite_files": false,
  "flush_report_interval": null
},
"Env":
{
  "env": "MountainCar-v0",
  "initial_reward": null,
  "final_reward": null,
  "min_reward": -1.0,
  "max_reward": 1.0,
  "render": true,
  "monitor": false,
  "monitor_dir": "./experiment-0/",
  "monitor_args":
  {
    "write_upon_reset": true,
    "video_callable": false
  }
},
"EnvRunner":
{
  "update_interval": 0.01,
  "inter_trial_duration": 0.4
},
"CommandReceiver":
{
  "socket": 5555,
  "time_stamp_tolerance": 0.01
},
"ObservationSender":
{
  "socket": 5556,
  "update_interval": 0.01
},
"RewardSender":
{
  "socket": 5557,
  "update_interval": 0.01
}
```

D.4 Example MUSIC configuration file

Listing D.3 shows an example MUSIC configuration file to run the MountainCar environment. It shows the different processes with parameters which are spawned by MUSIC including RMT adapters and NEST.

LISTING D.3: Example MUSIC configuration file to run the MountainCar environment.

```

stoptime=150.
rtf=1.
[reward]
  binary=zmq_in_adapter
  args=
  np=1
  music_timestep=0.001
  message_type=GymObservation
  zmq_topic=
  zmq_addr=tcp://localhost:5557
[sensor]
  binary=zmq_in_adapter
  args=
  np=1
  music_timestep=0.001
  message_type=GymObservation
  zmq_topic=
  zmq_addr=tcp://localhost:5556
[discretize]
  binary=discretize_adapter
  args=
  np=1
  music_timestep=0.001
  grid_positions_filename=grid_pos.json
[nest]
  binary=../actor_critic_network/network.py
  args=-t 150. -n 25 -m 3 -p network_params.json
  np=1
[argmax]
  binary=argmax_adapter
  args=
  np=1
  music_timestep=0.001
[command]
  binary=zmq_out_adapter
  args=
  np=1
  music_timestep=0.01
  message_type=GymCommand
  zmq_topic=
  zmq_addr=tcp://*:5555
sensor.out->discretize.in[2]
discretize.out->nest.in[25]
reward.out->nest.reward_in[1]
nest.out->argmax.in[3]
argmax.out->command.in[1]

```

D.5 Environments

Table D.4 shows parameters for the OpenAI Gym environments and the ZeroMQ wrapper.

OpenAI Gym	
Name	Values
Version	0.8.1
MountainCar	
Name	Values
Version	0
Max episode steps	None
Initial reward*	-1.0
Final reward*	-0.4
Inter-trial duration*	0.4
Update interval (env runner)*	0.02
FrozenLake	
Name	Values
Version	0
Max episode steps	None
Slippery	False
Final reward null*	-0.1
Inter-trial duration*	0.1
Update interval (env runner)*	0.1

TABLE D.4: Table of the environment parameters. Values marked with * indicate values for the ZeroMQ wrapper.

Bibliography

- Abadi, Martín et al. (2016). "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467*.
- Abbott, L. F. and P. Dayan (1999). "The Effect of Correlated Variability on the Accuracy of a Population Code". In: *Neural Comput.* 11, pp. 91–101.
- Abbott, LF, Brian DePasquale, and Raoul-Martin Memmesheimer (2016). "Building functional networks of spiking model neurons". In: *Nat. Neurosci.* 19.3, pp. 350–355.
- Abeles, Moshe (1991). *Corticonics: Neural Circuits of the Cerebral Cortex*. 1st. Cambridge: Cambridge University Press.
- Acerbi, Luigi, Sethu Vijayakumar, and Daniel M Wolpert (2014). "On the origins of suboptimality in human probabilistic inference". In: *PLoS Comput. Biol.* 10.6, e1003661.
- Ackley, David H, Geoffrey E Hinton, and Terrence J Sejnowski (1985). "A learning algorithm for Boltzmann machines". In: *Cog. Sci.* 9.1, pp. 147–169.
- Adhianto, Laksono et al. (2010). "HPCToolkit: Tools for performance analysis of optimized parallel programs". In: *Concurrency and Computation: Practice and Experience* 22.6, pp. 685–701.
- Aitchison, Laurence and Máté Lengyel (2016). "The Hamiltonian brain: efficient probabilistic inference with excitatory-inhibitory neural circuit dynamics". In: *PLoS Comput. Biol.* 12.12, e1005186.
- Albada, S. J. van et al. (2014). "Integrating brain structure and dynamics on supercomputers". In: *Brain-Inspired Computing*. Ed. by L. Grandinetti, T. Lippert, and N. Petkov. Springer, pp. 22–32.
- Albada, Sacha Jennifer van, Moritz Helias, and Markus Diesmann (2015). "Scalability of asynchronous networks is limited by one-to-one mapping between effective connectivity and correlations". In: *PLoS Comput. Biol.* 11.9, e1004490. DOI: 10.1371/journal.pcbi.1004490.
- Allen, Christina and Charles F Stevens (1994). "An evaluation of causes for unreliability of synaptic transmission". In: *Proc. Natl. Acad. Sci. USA* 91.22, pp. 10380–10383.
- Amit, D J (1988). "Neural networks counting chimes." In: *PNAS* 85 (7), pp. 2141–2145.
- Amit, Daniel J. and Nicolas Brunel (1997). "Model of Global Spontaneous Activity and Local Structured Activity During Delay periods in the Cerebral Cortex". In: *Cereb. Cortex* 7, pp. 237–252.
- Angelaki, Dora E, Yong Gu, and Gregory C DeAngelis (2009). "Multisensory integration: psychophysics, neurophysiology, and computation". In: *Curr. Opin. Neurobiol.* 19.4, pp. 452–458.
- Apolloni, Bruno and Diego de Falco (1991). "Learning by asymmetric parallel Boltzmann machines". In: *Neural Comput.* 3.3, pp. 402–408.
- Artola, Alain, S Brocher, and Wolf Singer (1990). "Different voltage-dependent thresholds for inducing long-term depression and long-term potentiation in slices of rat visual cortex". In: *Nature* 347.6288, p. 69.

- Aswolinskiy, Witali and Gordon Pipa (2015). "RM-SORN: a reward-modulated self-organizing recurrent neural network". In: *Frontiers in computational neuroscience* 9, p. 36.
- Averbeck, Bruno B., Peter E. Latham, and Alexandre Pouget (2006). "Neural correlations, population coding and computation". In: *natrevnsci* 7, pp. 358–366.
- Avermann, M et al. (2012). "Microcircuits of excitatory and inhibitory neurons in layer 2/3 of mouse barrel cortex". In: *J. Neurophysiol.* 107.11, pp. 3116–3134.
- Azevedo, Frederico A.C. et al. (2009). "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain". In: *J. Comp. Neurol.* 513.5, pp. 532–541.
- Badoni, D. et al. (2006). "An aVLSI recurrent network of spiking neurons with reconfigurable and plastic synapses". In: *Proceedings of the 2006 International Symposium on Circuits and Systems (ISCAS)*. Island of Kos: IEEE Press, p. 4.
- Bair, W., E. Zohary, and W.T. Newsome (2001). "Correlated firing in Macaque visual area MT: time scales and relationship to behavior". In: *J. Neurosci.* 21.5, pp. 1676–1697.
- Baladron, Javier and Fred H Hamker (2015). "A spiking neural network based on the basal ganglia functional anatomy". In: *Neural Networks* 67, pp. 1–13.
- Bavelier, Daphne and Helen J Neville (2002). "Cross-modal plasticity: where and how?" In: *Nature reviews. Neuroscience* 3.6, p. 443.
- Beck, Jeffrey M et al. (2008). "Probabilistic population codes for Bayesian decision making". In: *Neuron* 60.6, pp. 1142–1152.
- Beck, Jeffrey M et al. (2012). "Not noisy, just wrong: the role of suboptimal inference in behavioral variability". In: *Neuron* 74.1, pp. 30–39.
- Bekolay, Trevor et al. (2013). "Nengo: a Python tool for building large-scale functional brain models". In: *Frontiers in neuroinformatics* 7.
- Bellemare, Marc et al. (2015). "The arcade learning environment: An evaluation platform for general agents". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
- Benesty, Jacob et al. (2001). *Advances in Network and Acoustic Echo Cancellation*. 1st ed. Berlin, Germany: Springer. DOI: 10.1007/978-3-662-04437-7.
- Bengio, Yoshua et al. (2007). "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems* 19, p. 153.
- Benjamin, B.V. et al. (2014). "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations". In: *Proc. IEEE* 102.5, pp. 699–716. DOI: 10.1109/JPROC.2014.2313565.
- Bentley, Jon Louis (1982). *Writing efficient programs*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.
- Berger, James O (2013). *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media.
- Berkes, Pietro et al. (2011). "Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment". In: *Science* 331.6013, pp. 83–87.
- Bernacchia, Alberto and Xiao-Jing Wang (2013). "Decorrelation by recurrent inhibition in heterogeneous neural circuits". In: *Neural Comput.* 25.7, pp. 1732–1767.
- Bernacchia, Alberto et al. (2011). "A reservoir of time constants for memory traces in cortical neurons". In: *Nature neuroscience* 14.3, pp. 366–372.
- Bernardo, José-Miguel and Adrian F. Smith (2000). *Bayesian Theory*. reprint. Wiley series in probability and mathematical statistics. First publ. 1994. New York: Wiley.
- Bi, G. and M. Poo (2001). "Synaptic modification by correlated activity: Hebb's postulate revisited". In: *Annu. Rev. Neurosci.* 24, pp. 139–66.

- Biermann, Daniel (2016). "Probabilistic Computations in Neural Networks with Stochastic Synapses". Master Thesis. RWTH Aachen University.
- Bill, Johannes and Robert Legenstein (2014). "A compound memristive synapse model for statistical learning through STDP in spiking neural networks". In: *Front. Neurosci.* 8.
- Bill, Johannes et al. (2010). "Compensating inhomogeneities of neuromorphic VLSI devices via short-term synaptic plasticity". In: *Front. Comput. Neurosci.* 4.129.
- Binas, Jonathan, Giacomo Indiveri, and Michael Pfeiffer (2016). "Spiking analog VLSI neuron assemblies as constraint satisfaction problem solvers". In: *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on.* IEEE, pp. 2094–2097.
- Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.
- Blake, Randolph and Nikos K Logothetis (2002). "Visual competition". In: *Nature Reviews Neuroscience* 3.1, pp. 13–21.
- Blausen.com (2014). *Medical gallery of Blausen Medical* 2014.
- Boahen, Kwabena A. (2000). "Point-to-Point Connectivity Between Neuromorphic Chips Using Address Events". In: *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing* 47.5, pp. 416–434.
- Borsanyi, Szabolcs et al. (2010). "The QCD equation of state with dynamical quarks". In: *Journal of High Energy Physics* 2010.11, p. 77.
- Bos, Hannah, Markus Diesmann, and Moritz Helias (2016). "Identifying Anatomical Origins of Coexisting Oscillations in the Cortical Microcircuit". In: *PLoS Comput. Biol.* 12.10, pp. 1–34. DOI: 10.1371/journal.pcbi.1005132. URL: <http://dx.doi.org/10.1371/journal.pcbi.1005132>.
- Bos, Hannah et al. (2015). *NEST 2.10.0*. DOI: 10.5281/zenodo.44222. URL: <https://doi.org/10.5281/zenodo.44222>.
- Bower, James M. and David Beeman (2007). "GENESIS (simulation environment)". In: *Scholarpedia* 2.3, p. 1383.
- BrainScaleS (2014). *Project website*. Available at: <http://www.brainscales.eu>.
- Braitenberg, Valentin and Almut Schüz (1991). *Anatomy of the Cortex: Statistics and Geometry*. Berlin, Heidelberg, New York: Springer-Verlag. ISBN: 3-540-53233-1.
- Branco, Tiago and Kevin Staras (2009). "The probability of neurotransmitter release: variability and feedback control at single synapses". In: *Nat. Rev. Neurosci.* 10.5, p. 373.
- Brea, Johanni et al. (2016). "Prospective coding by spiking neurons". In: *PLoS Comput. Biol.* 12.6, e1005003.
- Bressler, Steven L and Vinod Menon (2010). "Large-scale brain networks in cognition: emerging methods and principles". In: *Trends in cognitive sciences* 14.6, pp. 277–290.
- Brette, R. et al. (2007). "Simulation of networks of spiking neurons: A review of tools and strategies". In: *J. Comput. Neurosci.* 23.3, pp. 349–398.
- Brette, Romain and Wulfram Gerstner (2005). "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity". In: *J. Neurophysiol.* 94.5, pp. 3637–3642.
- Brink, S. et al. (2013). "A Learning-Enabled Neuron Array IC Based Upon Transistor Channel Models of Biological Phenomena". In: *IEEE Trans. Biomed. Circuits Syst.* 7.1, pp. 71–81. DOI: 10.1109/TBCAS.2012.2197858.
- Broca, Paul (1865). "Sur le siège de la faculté du langage articulé". In: *Bull Soc Anthropol* 6, pp. 377–393.
- Brockman, G. et al. (2016). "OpenAI Gym". In: *ArXiv e-prints*. arXiv: 1606.01540 [cs.LG].

- Brodmann, Korbinian (1909). *Vergleichende Lokalisationslehre der Großhirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Leipzig: Johann Ambrosius Barth.
- Brüderle, Daniel et al. (2009). "Establishing a Novel Modeling Tool: A Python-based Interface for a Neuromorphic Hardware System". In: *Front. Neuroinformatics* 3.17.
- Brüderle, Daniel et al. (2010). "Simulator-Like Exploration of Cortical Network Architectures with a Mixed-Signal VLSI System". In: *Proceedings of the 2010 International Symposium on Circuits and Systems (ISCAS)*. Paris, France: IEEE Press.
- Brunel, Nicolas (2000). "Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons". In: *J. Comput. Neurosci.* 8.3, pp. 183–208.
- Brunel, Nicolas and Mark C. W. van Rossum (2007). "Lapicque's 1907 paper: from frogs to integrate-and-fire". In: *Biol. Cybern.* 97, pp. 337–339.
- Bryant, Hugh L and José P Segundo (1976). "Spike initiation by transmembrane current: a white-noise analysis." In: *J. Physiol.* 260.2, pp. 279–314.
- Buesing, Lars et al. (2011). "Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons". In: *PLoS Comput. Biol.* 7.11, e1002211.
- Buice, Michael A and Carson C Chow (2011). "Effective stochastic behavior in dynamical systems with incomplete information". In: *Phys. Rev. E* 84.5, p. 051120.
- Burkitt, Anthony N (2006a). "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input". In: *Biol. Cybern.* 95.1, pp. 1–19.
- (2006b). "A review of the integrate-and-fire neuron model: II. Inhomogeneous synaptic input and network properties". In: *Biological cybernetics* 95.2, pp. 97–112.
- Buzsáki, G and K Mizuseki (2014). "The log-dynamic brain: how skewed distributions affect network operations". In: *Nat. Rev. Neurosci.* 15, pp. 264–278.
- Buzsáki, György et al. (2015). "Tools for probing local circuits: high-density silicon probes combined with optogenetics". In: *Neuron* 86.1, pp. 92–105.
- Bytschok, Ilja et al. (2017). "Spike-based probabilistic inference with correlated noise". In: *arXiv preprint arXiv:1707.01746*.
- Cajal, Santiago Ramon y (1899). *Comparative study of the sensory areas of the human cortex*. Clark University.
- Camperi, Marcelo and Xiao-Jing Wang (1998). "A model of visuospatial working memory in prefrontal cortex: recurrent network and cellular bistability". In: *J. Comput. Neurosci.* 5.4, pp. 383–405.
- Cappello, Franck et al. (2014). "Toward exascale resilience: 2014 update". In: *Supercomputing frontiers and innovations* 1.1, pp. 5–28.
- Carnevale, Ted and Michael Hines (2006). *The NEURON Book*. Cambridge: Cambridge University Press.
- Cassidy, Andrew S et al. (2013). "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores". In: *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, pp. 1–10.
- Cavin, Ralph K, Paolo Lugli, and Victor V Zhirnov (2012). "Science and engineering beyond Moore's law". In: *Proceedings of the IEEE 100.Special Centennial Issue*, pp. 1720–1749.
- Chater, Nick, Joshua B. Tenenbaum, and Alan Yuille (2006). "Probabilistic models of cognition: Conceptual foundations". In: *Trends Cogn. Sci.* 10.7, pp. 287–291.
- Chelaru, Mircea I. and Valentin Dragoi (2008). "Efficient coding in heterogeneous neuronal populations". In: *Proc. Natl. Acad. Sci. USA* 105.42.
- Chua, Leon (1971). "Memristor-the missing circuit element". In: *IEEE Transactions on circuit theory* 18.5, pp. 507–519.
- Clopath, Claudia et al. (2010). "Connectivity reflects coding: a model of voltage-based STDP with homeostasis". In: *Nat. Neurosci.* 13, pp. 344–352.

- Cohen, Leonardo G et al. (1997). "Functional relevance of cross-modal plasticity in blind humans". In: *Nature* 389.6647, p. 180.
- Cohen, Marlene R and John H R Maunsell (2009). "Attention improves performance primarily by reducing interneuronal correlations". In: *Nat. Neurosci.* 12, pp. 1594–1600. DOI: 10.1038/nn.2439.
- Collobert, Ronan, Samy Bengio, and Johnny Mariéthoz (2002). *Torch: a modular machine learning software library*. Tech. rep. Idiap.
- Conklin, John and Chris Eliasmith (2005). "A controlled attractor network model of path integration in the rat". In: *Journal of computational neuroscience* 18.2, pp. 183–203.
- Coolen, A. C. C. (2001). "Statistical mechanics of recurrent neural networks I. Statistics". In: *Handbook of biological physics* 4, pp. 553–618.
- Cox, Richard T (1946). "Probability, frequency and reasonable expectation". In: *American Journal of Physics* 14.1, pp. 1–13.
- Crair, Michael C and William Bialek (1990). "Non-Boltzmann dynamics in networks of spiking neurons". In: *Advances in neural information processing systems*, pp. 109–116.
- Crick, Francis (1995). *The Astonishing Hypothesis: The Scientific Search for the Soul*. New York: Touchstone.
- Crick, Francis and Christof Koch (1990). "Towards a neurobiological theory of consciousness". In: *Seminars in the Neurosciences*. Vol. 2. Saunders Scientific Publications, pp. 263–275.
- Dahmen, David, Hannah Bos, and Moritz Helias (2016). "Correlated Fluctuations in Strongly Coupled Binary Networks Beyond Equilibrium". In: *Phys. Rev. X* 6 (3), p. 031024. DOI: 10.1103/PhysRevX.6.031024.
- DARPA (2008). *SyNAPSE*. URL: <https://www.darpa.mil/program/systems-of-neuromorphic-adaptive-plastic-scalable-electronics>.
- Davison, Andrew et al. (2009). "PyNN: a common interface for neuronal network simulators". In: *Front. Neuroinformatics* 2.11.
- Davison, Andrew P et al. (2008a). "PyNN: a common interface for neuronal network simulators". In: *Frontiers in neuroinformatics* 2.
- Davison, A.P. et al. (2008b). "PyNN: a common interface for neuronal network simulators". In: *Front. Neuroinformatics* 2.11. doi:10.3389/neuro.11.011.2008.
- Dayan, P. and L. F. Abbott (2001). *Theoretical Neuroscience*. Cambridge: MIT Press.
- de Kamps, Marc et al. (2008). "The state of MIIND". In: *Neural Networks* 21.8, pp. 1164–1181.
- De la Rocha, Jaime et al. (2007). "Correlation between neural spike trains increases with firing rate". In: *Nature* 448.16, pp. 802–807.
- Deco, Gustavo and Viktor K. Jirsa (2012). "Ongoing Cortical Activity at Rest: Criticality, Multistability, and Ghost Attractors". In: *J. Neurosci.* 32.10, pp. 3366–3375.
- Deco, Gustavo, Mario Senden, and Victor Jirsa (2012). "How anatomy shapes dynamics: a semi-analytical study of the brain at rest by a simple spin model". In: *Front. Comput. Neurosci.* 6.68. DOI: 10.3389/fncom.2012.00068.
- Delbruck, Tobias (1993). "Silicon retina with correlation-based, velocity-tuned pixels". In: *IEEE Transactions on Neural Networks* 4.3, pp. 529–541.
- Deng, J. et al. (2009). "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*.
- Denison, Stephanie et al. (2013). "Rational variability in children's causal inferences: The sampling hypothesis". In: *Cognition* 126.2, pp. 285–300.
- Denker, Michael et al. (2004). "Breaking Synchrony by Heterogeneity in Complex Networks". In: *Phys. Rev. Lett.* 92.7, pp. 074103–1–074103–4.

- Dere, E and A Zlomuzica (2011). "The role of gap junctions in the brain in health and disease." In: *Neurosci. Biobehav. Rev.* 36, pp. 206–217.
- Destexhe, Alain, Michael Rudolph, and Denis Pare (2003). "The high-conductance state of neocortical neurons in vivo". In: *Nat. Rev. Neurosci.* 4, pp. 739–751.
- Destexhe, Alain, Michael Rudolph, and Denis Paré (2003). "The high-conductance state of neocortical neurons in vivo". In: *Nature reviews neuroscience* 4.9, pp. 739–751.
- Diaz-Pier, Sandra et al. (2016). "Automatic Generation of Connectivity for Large-Scale Neuronal Network Models through Structural Plasticity". In: *Front. Neuroanatomy* 10, p. 57. DOI: 10.3389/fnana.2016.00057.
- Diesmann, Markus and Marc-Oliver Gewaltig (2002). "NEST: An Environment for Neural Systems Simulations". In: *Forschung und wissenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 2001*. Ed. by Theo Plesser and Volker Macho. Vol. 58. GWDG-Bericht. Göttingen: Ges. für Wiss. Datenverarbeitung, pp. 43–70.
- Diesmann, Markus, Marc-Oliver Gewaltig, and Ad Aertsen (1999). "Stable propagation of synchronous spiking in cortical neural networks". In: *Nature* 402.6761, pp. 529–533.
- Djurfeldt, Mikael, Örjan Ekeberg, and Anders Lansner (2008). "Large-scale modeling—a tool for conquering the complexity of the brain". In: *Front. Neuroinformatics* 2.1. doi:10.3389/neuro.11.001.2008.
- Djurfeldt, Mikael et al. (2010a). "Run-Time Interoperability Between Neuronal Network Simulators Based on the MUSIC Framework". In: *Neuroinformatics* 8, pp. 43–60.
- Djurfeldt, Mikael et al. (2010b). "Run-Time Interoperability between Neuronal Simulators based on the MUSIC Framework". In: *Neuroinformatics* 8 (43). doi:10.1007/s12021-010-9064-z.
- Dongarra, Jack et al. (2011). "The international exascale software project roadmap". In: *International Journal of High Performance Computing Applications* 25.1, pp. 3–60.
- Doya, Kenji (2000). "Reinforcement Learning in Continuous Time and Space". In: *Neural Comput.* 12.1, pp. 219–245.
- (2007). *Bayesian brain: Probabilistic approaches to neural coding*. MIT press.
- Dyson, George (2012). *Turing's cathedral: the origins of the digital universe*. Pantheon.
- Eccles, JC, Fatt P, and K Koketsu (1954). "Cholinergic and inhibitory synapses in a pathway from motor-axon collaterals to motoneurons". In: *J Physiol (Lond)* 126, pp. 524–562.
- Ecker, Alexander S. et al. (2010). "Decorrelated Neuronal Firing in Cortical Microcircuits". In: *Science* 327.5965, pp. 584–587.
- Ecker, Alexander S. et al. (2011). "The Effect of Noise Correlations in Populations of Diversely Tuned Neurons". In: *J. Neurosci.* 31.40, pp. 14272–14283.
- Eckhardt, Roger (1987). "Stan Ulam, John von Neumann, and the Monte Carlo Method". In: *Los Alamos Science* 15, pp. 131–137.
- Eliasmith, Chris and Charles H Anderson (2004). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press.
- Eppler, J. M. et al. (2009). "PyNEST: a convenient interface to the NEST simulator". In: *Front. Neuroinformatics* 2, p. 12. DOI: 10.3389/neuro.11.012.2008.
- Evans, Jason (2011). "Scalable memory allocation using jemalloc". In: *Notes by Facebook Engineering*.
- FACETS (2010). *Fast Analog Computing with Emergent Transient States*, project website. Available at: <http://www.facets-project.org>.

- Faisal, A Aldo and Simon B Laughlin (2007). "Stochastic simulations on the reliability of action potential propagation in thin axons". In: *PLoS Comput. Biol.* 3.5, e79.
- Faisal, A Aldo, Luc PJ Selen, and Daniel M Wolpert (2008a). "Noise in the nervous system". In: *Nature Reviews Neuroscience* 9.4, pp. 292–303.
- (2008b). "Noise in the nervous system". In: *Nature reviews. Neuroscience* 9.4, p. 292.
- Felleman, Daniel J and David C Van Essen (1991). "Distributed hierarchical processing in the primate cerebral cortex". In: *Cereb. Cortex* 1, pp. 1–47.
- Felleman, Daniel J and David C Van Essen (1991). "Distributed hierarchical processing in the primate cerebral cortex." In: *Cerebral cortex (New York, NY: 1991)* 1.1, pp. 1–47.
- Finetti, Bruno de (1990a). *Theory of Probability: A critical introductory treatment. Vol. 1.* Transl. by Antonio Machi and Adrian Smith; first publ. in Italian 1970. New York: Wiley.
- (1990b). *Theory of Probability: A critical introductory treatment. Vol. 2.* Transl. by Antonio Machi and Adrian Smith; first publ. in Italian 1970. New York: Wiley.
- Fiser, J. et al. (2010). "Statistically optimal perception and learning: from behavior to neural representations." In: *Trends Cogn. Sci.* 14, pp. 119–130.
- Fourcaud, Nicolas and Nicolas Brunel (2002). "Dynamics of the firing probability of noisy integrate-and-fire neurons". In: *Neural Comput.* 14, pp. 2057–2110.
- Frémaux, Nicolas and Wulfram Gerstner (2015). "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules". In: *Frontiers in neural circuits* 9.
- Frémaux, Nicolas, Henning Sprekeler, and Wulfram Gerstner (2010). "Functional Requirements for Reward-Modulated Spike-Timing-Dependent Plasticity". In: *J. Neurosci.* 30.40, pp. 13326–13337.
- Frémaux, Nicolas, Henning Sprekeler, and Wulfram Gerstner (2013). "Reinforcement learning using a continuous time actor-critic framework with spiking neurons". In: *PLoS Comput Biol* 9.4, e1003024.
- Friedrich, Johannes and Máté Lengyel (2016). "Goal-directed decision making with spiking neurons". In: *Journal of Neuroscience* 36.5, pp. 1529–1546.
- Friedrich, Johannes, Robert Urbanczik, and Walter Senn (2014). "Code-specific learning rules improve action selection by populations of spiking neurons". In: *International Journal of Neural Systems* 24.05, p. 1450002.
- Fries, Pascal (2005). "A mechanism for cognitive dynamics: neuronal communication through neuronal coherence". In: *Trends Cogn. Sci.* 9.10, pp. 474–480. DOI: 10.1016/j.tics.2005.08.011.
- Furber, S.B. et al. (2014). "The SpiNNaker Project". In: *Proc. IEEE* 102.5, pp. 652–665. DOI: 10.1109/JPROC.2014.2304638.
- Furber, Steve (2016). "Large-scale neuromorphic computing systems". In: *Journal of neural engineering* 13.5, p. 051001.
- Furber, Steve B et al. (2013). "Overview of the spinnaker system architecture". In: *IEEE Transactions on Computers* 62.12, pp. 2454–2467.
- Gantmacher, F. R. (1990). *The Theory of Matrices, Vol. 1.* 2nd ed. American Mathematical Society.
- Gardiner, Crispin (1985). *Stochastic methods.* Springer Series in Synergetics. Springer.
- Gelman, Andrew et al. (2014). *Bayesian data analysis.* Vol. 2. CRC press Boca Raton, FL.
- Geman, Stuart and Donald Geman (1984a). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6.6, pp. 721–741.

- Geman, Stuart and Donald Geman (1984b). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6.6, pp. 721–741.
- Gershman, Samuel J, Eric J Horvitz, and Joshua B Tenenbaum (2015). "Computational rationality: A converging paradigm for intelligence in brains, minds, and machines". In: *Science* 349.6245, pp. 273–278.
- Gershman, Samuel J, Edward Vul, and Joshua B Tenenbaum (2012). "Multistability and perceptual inference". In: *Neural computation* 24.1, pp. 1–24.
- Gerstner, W. and W. Kistler (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press. ISBN: (paperback) 0521890799.
- Gerstner, Wulfram and Richard Naud (2009). "How good are neuron models?" In: *Science* 326.5951, pp. 379–380.
- Gewaltig, Marc-Oliver and Markus Diesmann (2007). "NEST (NEural Simulation Tool)". In: *Scholarpedia* 2.4, p. 1430. DOI: 10.4249/scholarpedia.1430.
- Giacomazzo, Bruno et al. (2012). "General relativistic simulations of magnetized plasmas around merging supermassive black holes". In: *The Astrophysical Journal Letters* 752.1, p. L15.
- Gierer, Alfred and Hans Meinhardt (1974). "Biological pattern formation involving lateral inhibition". In: *Lectures on mathematics in the life sciences* 7, p. 163.
- Ginis, George and Chia-Ning Peng (2006). "Alien crosstalk cancellation for multi-pair digital subscriber line systems". In: *EURASIP Journal on Advances in Signal Processing* 2006.1, pp. 1–12.
- Ginzburg, Iris and Haim Sompolinsky (1994). "Theory of correlations in stochastic neural networks". In: *Phys. Rev. E* 50.4, p. 3171.
- Giulioni, Massimiliano et al. (2012). "Robust working memory in an asynchronously spiking neural network realized in neuromorphic VLSI". In: *Front. Neurosci.* 5.149. ISSN: 1662-453X. DOI: 10.3389/fnins.2011.00149. URL: http://www.frontiersin.org/neuromorphic_engineering/10.3389/fnins.2011.00149/abstract.
- Gleeson, Pdraig et al. (2010). "NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail". In: *PLoS Comput. Biol.* 6.6, e1000815.
- Golomb, David and John Rinzel (1993). "Dynamics of globally coupled inhibitory neurons with heterogeneity". In: *Phys. Rev. E* 48.6, pp. 4810–4814.
- Good, Irving John (1950). *Probability and the Weighing of Evidence*. London: Charles Griffin & Company.
- (1983). *Good Thinking: The Foundations of Probability and Its Applications*. Minneapolis, USA: University of Minnesota Press.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press.
- Goodman, D. F. M. and R. Brette (2009). "The Brian simulator". In: *Front. Neurosci.* 3.2. doi: 10.3389/neuro.01.026.2009, pp. 192–197.
- Goodman, D.F.M. and R. Brette (2013). "Brian Simulator". In: *Scholarpedia* 8.1, p. 10883.
- Goulas, Alexandros, Alexander Schaefer, and Daniel S Margulies (2015). "The strength of weak connections in the macaque cortico-cortical network". In: *Brain Structure and Function* 220.5, pp. 2939–2951.
- Griffith, J. S. and G. Horn (1966). "An analysis of spontaneous impulse activity of units in the striate cortex of unrestrained cats". In: *J. Physiol.* 186.3, pp. 516–534.
- Griffiths, Thomas L and Joshua B Tenenbaum (2006). "Optimal predictions in everyday cognition". In: *Psychological science* 17.9, pp. 767–773.

- Griffiths, Thomas L et al. (2010). "Probabilistic models of cognition: Exploring representations and inductive biases". In: *Trends in cognitive sciences* 14.8, pp. 357–364.
- Grytskyy, D. et al. (2013a). "A unified view on weakly correlated recurrent networks". In: *Front. Comput. Neurosci.* 7, p. 131. DOI: 10.3389/fncom.2013.00131.
- Grytskyy, Dmytro et al. (2013b). "A unified view on weakly correlated recurrent networks". In: *Frontiers in computational neuroscience* 7.
- Gütig, Robert and Haim Sompolinsky (2006). "The tempotron: a neuron that learns spike timing-based decisions". In: *Nature neuroscience* 9.3, p. 420.
- Habenschuss, Stefan, Zeno Jonke, and Wolfgang Maass (2013). "Stochastic computations in cortical microcircuit models". In: *PLoS Comput. Biol.* 9.11, e1003311.
- Haefner, Ralf M, Pietro Berkes, and József Fiser (2016). "Perceptual decision-making as probabilistic inference by neural sampling". In: *Neuron* 90.3, pp. 649–660.
- Hafliger, P. (2007). "Adaptive WTA With an Analog VLSI Neuromorphic Learning Chip". In: *IEEE Trans. Neural Netw.* 18.2, pp. 551–572.
- Hagen, Espen et al. (2016). "Hybrid Scheme for Modeling Local Field Potentials from Point-Neuron Networks". In: *Cereb. Cortex*, bhw237. DOI: 10.1093/cercor/bhw237.
- Hager, Georg and Gerhard Wellein (2011). *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton, FL: CRC Press. ISBN: 978-1-4398-1192-4.
- Hahne, J et al. (2015). "A unified framework for spiking and gap-junction interactions in distributed neuronal network simulations." In: *Front. Neuroinformatics* 9.22. DOI: 10.3389/fninf.2015.00022.
- Hahne, J et al. (2016). "Integration of continuous-time dynamics in a spiking neural network simulator". In: *arXiv*. 1610.09990 [q-bio.NC].
- (2017). "Integration of continuous-time dynamics in a spiking neural network simulator". In: *Front. Neuroinformatics* 11, p. 34. ISSN: 1662-5196. DOI: 10.3389/fninf.2017.00034. URL: <http://journal.frontiersin.org/article/10.3389/fninf.2017.00034>.
- Hamid, Nor Hisham, Tong Boon Tang, and Alan F Murray (2011). "Probabilistic neural computing with advanced nanoscale MOSFETs". In: *Neurocomputing* 74.6, pp. 930–940.
- Hansen, Enrique CA et al. (2015). "Functional connectivity dynamics: modeling the switching behavior of the resting state". In: *Neuroimage* 105, pp. 525–535.
- Hardingham, Neil R and Alan U Larkman (1998). "The reliability of excitatory synaptic transmission in slices of rat visual cortex in vitro is temperature dependent". In: *The Journal of Physiology* 507.1, pp. 249–256.
- Harris, Kenneth D. and Alexander Thiele (2011). "Cortical state and attention". In: *Nat. Rev. Neurosci.* 12, pp. 509–523. DOI: 10.1038/nrn3084.
- Hastings, W Keith (1970). "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1, pp. 97–109.
- Hawkins, Jeff and Sandra Blakeslee (2004). *On Intelligence*. New York: St. Martin's Griffin.
- Hebb, Donald Olding (1967). *The Organization of Behaviour: A Neuropsychological Theory*. John Wiley & Sons.
- Helias, Moritz, Tom Tetzlaff, and Markus Diesmann (2013). "Echoes in correlated neural systems". In: *New J. Phys.* 15, p. 023002.
- (2014a). "The correlation structure of local cortical networks intrinsically results from recurrent dynamics". In: *PLoS Comput. Biol.* 10.1, e1003428.

- Helias, Moritz, Tom Tetzlaff, and Markus Diesmann (2014b). "The correlation structure of local cortical networks intrinsically results from recurrent dynamics". In: *PLoS Comput Biol* 10.1, e1003428.
- Helias, Moritz et al. (2012). "Supercomputers ready for use as discovery machines for neuroscience". In: *Front. Neuroinform.* 6, p. 26. DOI: 10.3389/fninf.2012.00026.
- Hellwig, Bernhard (2000). "A quantitative analysis of the local connectivity between pyramidal neurons in layers 2/3 of the rat visual cortex". In: *Biol. Cybern.* 2.82, pp. 111–121.
- Hennequin, Guillaume, Tim P Vogels, and Wulfram Gerstner (2014). "Optimal control of transient dynamics in balanced networks supports generation of complex movements". In: *Neuron* 82.6, pp. 1394–1406.
- Herculano-Houzel, Suzana (2011). "Scaling of brain metabolism with a fixed energy budget per neuron: implications for neuronal activity, plasticity and evolution". In: *PloS one* 6.3, e17514.
- (2012). "The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost". In: *Proceedings of the National Academy of Sciences* 109.Supplement 1, pp. 10661–10668.
- Hertz, John, Anders Krogh, and Richard G. Palmer (1991). *Introduction to the Theory of Neural Computation*. Perseus Books. ISBN: 0-201-51560-1.
- Hilpert, Simon et al. (2017). "Addressing energy system modelling challenges: The contribution of the Open Energy Modelling Framework (oemof)". In:
- Hines, M., S. Kumar, and F. Schürmann (2011). "Comparison of neuronal spike exchange methods on a Blue Gene/P supercomputer". In: *Front. Comput. Neurosci.* 5.49. 10.3389/fncom.2011.00049.
- Hines, Michael L and N Ted Carnevale (2008). "Translating network models to parallel hardware in NEURON". In: *Journal of neuroscience methods* 169.2, pp. 425–455.
- Hintjens, Pieter (2013). *ZeroMQ: Messaging for Many Applications*. "O'Reilly Media, Inc."
- Hinton, G. E. and R. R. Salakhutdinov (2006a). "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786, pp. 504–507.
- Hinton, G E, T J Sejnowski, and D H Ackley (1984a). *Boltzmann machines: constraint satisfaction networks that learn*. Technical Report CMU-CS-84-119. Carnegie-Mellon University.
- Hinton, Geoffrey (2010). "A practical guide to training restricted Boltzmann machines". In: *Momentum* 9.1, p. 926.
- Hinton, Geoffrey et al. (2012a). "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal Processing Magazine* 29.6, pp. 82–97.
- Hinton, Geoffrey E (2002). "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14.8, pp. 1771–1800.
- Hinton, Geoffrey E, Simon Osindero, and Yee-Whye Teh (2006). "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7, pp. 1527–1554.
- Hinton, Geoffrey E and Ruslan R Salakhutdinov (2006b). "Reducing the dimensionality of data with neural networks". In: *science* 313.5786, pp. 504–507.
- Hinton, Geoffrey E, Terrence J Sejnowski, and David H Ackley (1984b). *Boltzmann machines: Constraint satisfaction networks that learn*. Tech. rep. Carnegie-Mellon University, Pittsburgh, PA.
- Hinton, Geoffrey E et al. (2012b). "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580*.

- Hodgkin, A. L. and A. F. Huxley (1952). "A Quantitative Description of Membrane Current and Its Application to Conduction and Excitation in Nerve". In: *J. Physiol.* 117, pp. 500–544.
- Holmstrom, Lars A et al. (2010). "Efficient encoding of vocalizations in the auditory midbrain". In: *J. Neurosci.* 30.3, pp. 802–819.
- Holt, Gary R et al. (1996). "Comparison of discharge variability in vitro and in vivo in cat visual cortex neurons". In: *Journal of neurophysiology* 75.5, pp. 1806–1814.
- Holtmaat, Anthony and Karel Svoboda (2009). "Experience-dependent structural synaptic plasticity in the mammalian brain". In: *Nature reviews. Neuroscience* 10.9, p. 647.
- Hopfield, J. J. (1982a). "Neural networks and physical systems with emergent collective computational abilities". In: *Proc. Natl. Acad. Sci. USA* 79, pp. 2554–2558.
- Hopfield, John J (1982b). "Neural networks and physical systems with emergent collective computational abilities". In: *Proc. Natl. Acad. Sci. USA* 79.8, pp. 2554–2558.
- Hornschild, Aaron (2014). "Neural Computation with Stochastic Synapses". Bachelor Thesis. RWTH Aachen University.
- Hoyer, Patrik O and Aapo Hyvarinen (2003). "Interpreting neural response variability as Monte Carlo sampling of the posterior". In: *Advances in neural information processing systems*, pp. 293–300.
- Hoyer, Patrik O and Aapo Hyvärinen (2003). "Interpreting neural response variability as Monte Carlo sampling of the posterior". In: *Advances in neural information processing systems*, pp. 293–300.
- Hromádka, Tomas, Michael R DeWeese, and Anthony M Zador (2008). "Sparse Representation of Sounds in the Unanesthetized Auditory Cortex". In: *PLoS Biol.* 6.1.
- Human Brain Project (2014). *Project website*. Available at: <http://www.humanbrainproject.eu>.
- Ikegaya, Yuji et al. (2013). "Interpyramid Spike Transmission Stabilizes the Sparseness of Recurrent Network Activity". In: *Cereb. Cortex* 23.2, pp. 293–304.
- Indiveri, Giacomo, Elisabetta Chicca, and Rodney Douglas (2009). "Artificial Cognitive Systems: From VLSI Networks of Spiking Neurons to Neuromorphic Cognition". In: *Cognitive Computation* 1.2, pp. 119–127. DOI: 10.1007/s12559-008-9003-6.
- Indiveri, Giacomo et al. (2011a). "Neuromorphic silicon neuron circuits". In: *Front. Neurosci.* 5.73. DOI: 10.3389/fnins.2011.00073.
- Indiveri, Giacomo et al. (2011b). "Neuromorphic silicon neuron circuits". In: *Frontiers in neuroscience* 5, p. 73.
- Ippen, Tammo et al. (2017). "Constructing neuronal network models in massively parallel environments". In: *Frontiers in Neuroinformatics* 11, p. 30.
- Izhikevich, Eugene M. (2007). "Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling". In: *Cereb. Cortex* 17.10, pp. 2443–2452.
- Jaeger, H. (2001). *The "echo state" approach to analysing and training recurrent neural networks*. Tech. rep. GMD Report 148. St. Augustin, Germany: German National Research Center for Information Technology.
- James, Bergstra et al. (2010). "Theano: a CPU and GPU math expression compiler". In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*.
- Jastrow, J. (1899). "The mind's eye". In: *Popular Science Monthly* 54, pp. 299–312.
- Jaynes, Edwin T (2003a). *Probability theory: The logic of science*. Cambridge university press.
- (2003b). *Probability theory: The logic of science*. Cambridge: Cambridge university press.

- Jaynes, ET (1988). "How does the brain do plausible reasoning?" In: *Maximum-entropy and Bayesian methods in science and engineering*. Springer, pp. 1–24.
- Jeffress, Lloyd A (1948). "A place theory of sound localization." In: *Journal of comparative and physiological psychology* 41.1, p. 35.
- Jeffreys, Harold (1973). *Scientific inference*. Cambridge University Press.
- Jia, Yangqing et al. (2014). "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093*.
- Jitsev, Jenia, Abigail Morrison, and Marc Tittgemeyer (2012). "Learning from positive and negative rewards in a spiking neural network model of basal ganglia". In: *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE, pp. 1–8.
- Johnson, Claes (1987). *Numerical solution of partial differential equations by the finite element method*. Cambridge: Cambridge University Press.
- Jonas, Eric and Konrad Paul Kording (2017). "Could a neuroscientist understand a microprocessor?" In: *PLoS computational biology* 13.1, e1005268.
- Jordan, Jakob, Philipp Weidel, and Abigail Morrison (2017). "Closing the loop between neural network simulators and the OpenAI Gym". In: *ArXiv e-prints*. arXiv: 1709.05650 [q-bio.NC].
- Jordan, Jakob et al. (2017a). "Extremely scalable spiking neural network simulation code: from laptops to exascale computers". (submitted).
- Jordan, Jakob et al. (2017b). "Stochastic neural computation without noise". In: *ArXiv e-prints*. arXiv: 1710.04931 [q-bio.NC].
- Jülich Supercomputing Centre (2015). "JUQUEEN: IBM Blue Gene/Q Supercomputer System at the Jülich Supercomputing Centre". In: *Journal of large-scale research facilities* 1. URL: <http://dx.doi.org/10.17815/jlsrf-1-18>.
- Kahneman, Daniel (2011). *Thinking, fast and slow*. Macmillan.
- Kahneman, Daniel, Paul Slovic, and Amos Tversky (1982). *Judgments under uncertainty: heuristics and biases*. Cambridge University Press.
- Kandel, Eric R., James H. Schwartz, and Thomas M. Jessel (1991). *Principles of neural science*. 3rd ed. New York: Elsevier. ISBN: 0-444-01562-0.
- (2000). *Principles of Neural Science*. 4th ed. ISBN 978-0838577011. New York: McGraw-Hill.
- Kandel, Eric R et al. (2013). "Neuroscience thinks big (and collaboratively)". In: *Nat. Rev. Neurosci.* 14.9, pp. 659–664.
- Kaplan, B. et al. (2009). "High-conductance states on a neuromorphic hardware system". In: *Proceedings of the 2009 International Joint Conference on Neural Networks (IJCNN)*. Atlanta: IEEE Press, pp. 1524–1530.
- Kappel, David et al. (2015). "Network plasticity as Bayesian inference". In: *PLoS Comput. Biol.* 11.11, e1004485.
- Kelly, Frank P (2011). *Reversibility and stochastic networks*. Cambridge University Press.
- Kenet, Tal et al. (2003). "Spontaneously emerging cortical representations of visual attributes". In: *Nature* 425.6961, p. 954.
- Kersten, Daniel, Pascal Mamassian, and Alan Yuille (2004). "Object perception as Bayesian inference". In: *Annu. Rev. Psychol.* 55, pp. 271–304.
- Kikombo, Andrew Kilinga et al. (2009). "A pulse-density modulation circuit exhibiting noise shaping with single-electron neurons". In: *Proceedings of the International Joint Conference on Neural Networks 2009 (IJCNN 2009)*. IEEE, pp. 1600–1605.
- Knill, David C and Whitman Richards (1996). *Perception as Bayesian inference*. Cambridge University Press.
- Knill, DC and A Pouget (2004). "The Bayesian brain: the role of uncertainty in neural coding and computation". In: *Trends Neurosci.* 27.12, pp. 712–719.

- Koch, K. W. and J. M. Fuster (1989). "Unit activity in monkey parietal cortex related to haptic perception and temporary memory". In: *Exp. Brain Res.* 76.2, pp. 292–306.
- Kock, Christiaan P. J. de and Bert Sakmann (2009). "Spiking in primary somatosensory cortex during natural whisking in awake head-restrained rats is cell-type specific". In: *Proc. Natl. Acad. Sci. USA* 106.38, pp. 16446–16450.
- Kohn, Adam and Matthew A. Smith (2005). "Stimulus dependence of neuronal correlations in primary visual cortex of the Macaque". In: *J. Neurosci.* 25.14, pp. 3661–3673.
- Körding, Konrad (2007). "Decision theory: what" should" the nervous system do?" In: *Science* 318.5850, pp. 606–610.
- Kording, Konrad P and Daniel M Wolpert (2004). "Bayesian integration in sensorimotor learning". In: *Nature* 427.6971, p. 244.
- Körding, Konrad P et al. (2004). "A neuroeconomics approach to inferring utility functions in sensorimotor control". In: *PLoS biology* 2.10, e330.
- Körding, Konrad P et al. (2007). "Causal inference in multisensory perception". In: *PLoS one* 2.9, e943.
- Koulakov, Alexei A., Tomas Hromádka, and Anthony M. Zador (2009). "Correlated Connectivity and the Distribution of Firing Rates in the Neocortex". In: *J. Neurosci.* 29.12, pp. 3685–3694.
- Kozloski, James and John Wagner (2011). "An ultrascaleable solution to large-scale neural tissue simulation". In: *Front. Neuroinform.* 5.15. DOI: 10.3389/fninf.2011.00015.
- Krauth, Werner (2006). *Statistical mechanics: algorithms and computations*. Vol. 13. Oxford Master Series in Physics. Oxford University Press.
- Kriener, Birgit et al. (2008). "Correlations and population dynamics in cortical networks". In: *Neural Comput.* 20, pp. 2185–2226.
- Krizhevsky, Alex and Geoffrey Hinton (2009). *Learning multiple layers of features from tiny images*. Tech. rep.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.
- Kuhn, A., A. Aertsen, and S. Rotter (2004). "Neuronal integration of synaptic input in the fluctuation-driven regime". In: *J. Neurosci.* 24.10, pp. 2345–2356.
- Kunkel, Susanne and Wolfram Schenck (2017a). "The NEST dry-run mode: Efficient dynamic analysis of neuronal network simulation code". In: *Front. Neuroinformatics* 11, p. 40.
- (2017b). "The NEST dry-run mode: Efficient dynamic analysis of neuronal network simulation code". In: *Frontiers in neuroinformatics* 11.
- Kunkel, Susanne et al. (2012a). "Meeting the memory challenges of brain-scale simulation". In: *Front. Neuroinformatics* 5, p. 35.
- Kunkel, Susanne et al. (2012b). "Memory Consumption of Neuronal Network Simulators at the Brain Scale". In: *NIC Symposium 2012 Proceedings*. Ed. by Kurt Binder, Gernot Münster, and Manfred Kremer. Vol. 45. NIC Series. Forschungszentrum Jülich GmbH, pp. 81–88.
- Kunkel, Susanne et al. (2014). "Spiking network simulation code for petascale computers". In: *Front. Neuroinformatics* 8, p. 78. ISSN: 1662-5196. DOI: 10.3389/fninf.2014.00078. URL: <http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2014.00078/abstract>.
- Kunkel, Susanne et al. (2017). *NEST 2.12.0*. DOI: 10.5281/zenodo.259534. URL: <https://doi.org/10.5281/zenodo.259534>.

- Lansner, Anders and Markus Diesmann (2012a). "Virtues, pitfalls, and methodology of neuronal network modeling and simulations on supercomputers". In: *Computational Systems Neurobiology*. Springer, pp. 283–315.
- (2012b). "Virtues, pitfalls, and methodology of neuronal network modeling and simulations on supercomputers". In: *Computational Systems Neurobiology*. Ed. by Nicolas Le Novère. Dordrecht: Springer. Chap. 10, pp. 283–315.
- Lapicque, L. (1907). "Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation". In: *J. Physiol. Pathol. Gen.* 9, pp. 620–635.
- Laplace (Marquis de), Pierre Simon (1820). *Théorie analytique des probabilités*. 3rd ed. Paris: Courcier.
- Le Roux, Nicolas and Yoshua Bengio (2008). "Representational power of restricted Boltzmann machines and deep belief networks". In: *Neural computation* 20.6, pp. 1631–1649.
- LeCun, Yann (1998). *The MNIST database of handwritten digits*.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.
- LeCun, Yann, Corinna Cortes, and Christopher JC Burges (1998). *The MNIST database of handwritten digits*.
- Lee, Jun Haeng, Tobi Delbruck, and Michael Pfeiffer (2016). "Training deep spiking neural networks using backpropagation". In: *Frontiers in neuroscience* 10.
- Lee, Tai Sing and David Mumford (2003). "Hierarchical Bayesian inference in the visual cortex". In: *JOSA A* 20.7, pp. 1434–1448.
- Lefort, Sandrine et al. (2009). "The Excitatory Neuronal Network of the C2 Barrel Column in Mouse Primary Somatosensory Cortex". In: *Neuron* 61, pp. 301–316.
- Legenstein, Robert and Wolfgang Maass (2014). "Ensembles of spiking neurons with noise support optimal probabilistic inference in a dynamically changing environment". In: *PLoS Comput. Biol.* 10.10, e1003859.
- Lengler, Johannes, Florian Jug, and Angelika Steger (2013). "Reliable Neuronal Systems: The Importance of Heterogeneity". In: *PLoS One*.
- Levy, William B and Robert A Baxter (2002). "Energy-efficient neuronal computation via quantal synaptic failures". In: *Journal of Neuroscience* 22.11, pp. 4746–4755.
- Li, Yi et al. (2013). "Ultrafast synaptic events in a chalcogenide memristor". In: *Scientific reports* 3.
- Li, Zhaoping and Peter Dayan (1999). "Computational differences between asymmetrical and symmetrical networks". In: *Advances in Neural Information Processing Systems*, pp. 274–280.
- Lieder, Falk, Thomas Griffiths, and Noah Goodman (2012). "Burn-in, bias, and the rationality of anchoring". In: *Advances in neural information processing systems*, pp. 2690–2798.
- Lindén, Henrik et al. (2011). "Modeling the spatial reach of the LFP". In: *Neuron* 72.5, pp. 859–872.
- Lindley, Dennis V. (2014). *Understanding Uncertainty*. rev. ed. Wiley series in probability and statistics. First publ. 2006. Hoboken, USA: Wiley.
- Litwin-Kumar, Ashok and Brent Doiron (2012). "Slow dynamics and high variability in balanced cortical networks with clustered connections". In: *Nat. Neurosci.* Pp. 1498–1505.
- Lloyd, S (2000). "Ultimate physical limits to computation". In: *Nature* 406.6799, p. 1047.
- Locke, Rachel, Jean Vautrin, and Stephen Highstein (1999). "Miniature EPSPs and sensory encoding in the primary afferents of the vestibular lagena of the toadfish, *Opsanus tau*". In: *Annals of the New York Academy of Sciences* 871.1, pp. 35–50.

- London, Michael et al. (2010). "Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex". In: *Nature* 466.1, pp. 123–128. DOI: 10.1038/nature09086.
- Lundqvist, Mikael, Albert Compte, and Anders Lansner (2010). "Bistable, Irregular Firing and Population Oscillations in a Modular Attractor Memory Network". In: *PLoS Comput. Biol.* 6.6, e1000803.
- Lundqvist, Mikael et al. (2006a). "Attractor Dynamics in a Modular Network Model of Neocortex". In: *Network: Comput. Neural Systems* 17.3, pp. 253–276.
- (2006b). "Attractor dynamics in a modular network model of neocortex". In: *Network: Computation in Neural Systems* 17.3, pp. 253–276.
- Ly, C, JW Middleton, and B Doiron (2012). "Cellular and circuit mechanisms maintain low spike co-variability and enhance population coding in somatosensory cortex". In: *Front. Comput. Neurosci.* 6.7, pp. 1–25.
- Lytton, William W et al. (2016). "Simulation neurotechnologies for advancing brain research: parallelizing large networks in NEURON". In: *Neural computation*.
- Ma, Wei Ji (2012). "Organizing probabilistic models of perception". In: *Trends in cognitive sciences* 16.10, pp. 511–518.
- Ma, Wei Ji and Mehrdad Jazayeri (2014). "Neural coding of uncertainty and probability". In: *Annual review of neuroscience* 37, pp. 205–220.
- Ma, Wei Ji et al. (2006). "Bayesian inference with probabilistic population codes". In: *Nature neuroscience* 9.11, pp. 1432–1438.
- Maass, Wolfgang (1997). "Networks of spiking neurons: The third generation of neural network models". In: *Neural Networks* 10.9, pp. 1659–1671. DOI: 10.1016/S0893-6080(97)00011-7.
- (2014). "Noise as a resource for computation and learning in networks of spiking neurons". In: *Proceedings of the IEEE* 102.5, pp. 860–880.
- Maass, Wolfgang, Thomas Natschläger, and Henry Markram (2002). "Real-time computing without stable states: A new framework for neural computation based on perturbations". In: *Neural computation* 14.11, pp. 2531–2560.
- Maass, Wolfgang, Thomas Natschläger, and Henry Markram (2002). "Real-time computing without stable states: a new framework for neural computation based on perturbation". In: *Neural Comput.* 14.11, pp. 2531–2560.
- MacKay, David JC (2003). *Information theory, inference and learning algorithms*. Cambridge university press.
- Mainen, Zachary F. and Terrence J. Sejnowski (1995a). "Reliability of Spike Timing in Neocortical Neurons". In: *Science* 268, pp. 1503–1506.
- (1995b). "Reliability of Spike Timing in Neocortical Neurons". In: *Science* 268, pp. 1503–1506.
- Major, Guy, Matthew E Larkum, and Jackie Schiller (2013). "Active properties of neocortical pyramidal neuron dendrites". In: *Annual review of neuroscience* 36, pp. 1–24.
- Mar, D.J. et al. (1999). "Noise shaping in populations of coupled model neurons". In: *Proc. Natl. Acad. Sci. USA* 96.18, 10450–10455.
- Marder, Eve and Jean-Marc Goaillard (2006). "Variability, compensation and homeostasis in neuron and network function". In: *Nat. Rev. Neurosci.* 7.7, pp. 563–574.
- Marienburg, Jan-Tore, Tor Sverre Lande, and M Hovén (2002). "Neuromorphic noise shaping in coupled neuron populations". In: *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*. Vol. 5. IEEE, pp. 73–76.
- Markov, N. T. et al. (2014a). "A Weighted and Directed Interareal Connectivity Matrix for Macaque Cerebral Cortex". In: *Cereb. Cortex* 24.1, pp. 17–36. DOI: 10.1093/cercor/bhs270. eprint: <http://cercor.oxfordjournals>.

- org/content/24/1/17.full.pdf+html. URL: <http://cercor.oxfordjournals.org/content/24/1/17.abstract>.
- Markov, Nikola T. et al. (2014b). "Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex". In: *J. Comp. Neurol.* 522.1, pp. 225–259. ISSN: 1096-9861. DOI: 10.1002/cne.23458. URL: <http://dx.doi.org/10.1002/cne.23458>.
- Markov, Nikola T et al. (2014c). "Anatomy of hierarchy: feedforward and feedback pathways in macaque visual cortex". In: *Journal of Comparative Neurology* 522.1, pp. 225–259.
- Markram, Henry et al. (2015). "Reconstruction and simulation of neocortical micro-circuitry". In: *Cell* 163.2, pp. 456–492.
- Marr, David and Tomaso Poggio (1976). "From understanding computation to understanding neural circuitry". In: *A.I. Memo* 357.
- Marsat, Gary and Leonard Maler (2010). "Neural Heterogeneity and Efficient Population Codes for Communication Signals". In: *J. Neurophysiol.* 104.5, pp. 2543–2555. ISSN: 0022-3077. DOI: 10.1152/jn.00256.2010.
- May, Arne (2011). "Experience-dependent structural plasticity in the adult human brain". In: *Trends in cognitive sciences* 15.10, pp. 475–482.
- McCulloch, W. S. and W. Pitts (1943). "A logical calculus of the ideas immanent in neural nets". In: *Bull. Math. Biophys.* 5, pp. 115–137.
- Mead, C. (1990). "Neuromorphic electronic systems". In: *Proc. IEEE* 78.10, pp. 1629–1636.
- Mead, Carver (1989). *Analog VLSI and neural systems*. Boston, MA, USA: Addison-Wesley.
- Mejias, J. F. and A. Longtin (2012). "Optimal Heterogeneity for Coding in Spiking Neural Networks". In: *Phys. Rev. Lett.* 108.
- Mejias, Jorge F and André Longtin (2014). "Differential effects of excitatory and inhibitory heterogeneity on the gain and asynchronous state of sparse cortical networks". In: *Frontiers in computational neuroscience* 8.
- Mendat, Daniel R et al. (2016). "Neuromorphic sampling on the SpiNNaker and parallella chip multiprocessors". In: *Circuits & Systems (LASCAS), 2016 IEEE 7th Latin American Symposium on*. IEEE, pp. 399–402.
- Merolla, Paul, Tristan Ursell, and John Arthur (2010). "The thermodynamic temperature of a rhythmic spiking network". In: *arXiv preprint arXiv:1009.5473*.
- Merolla, Paul A. et al. (2014). "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197, pp. 668–673. DOI: 10.1126/science.1254642.
- Message Passing Interface Forum (2009). *MPI: A Message-Passing Interface Standard, Version 2.2*. URL: <http://www.mpi-forum.org/docs/>.
- Metropolis, Nicholas and Stanislaw Ulam (1949). "The monte carlo method". In: *Journal of the American statistical association* 44.247, pp. 335–341.
- Metropolis, Nicholas et al. (1953). "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6, pp. 1087–1092.
- Middleton, JW et al. (2012). "Neural correlation is stimulus modulated by feedforward inhibitory circuitry". In: *J. Neurosci.* 32.2, pp. 506–518.
- Migliore, Michele et al. (2006). "Parallel network simulations with NEURON". In: *Journal of computational neuroscience* 21.2, pp. 119–129.
- Millner, Sebastian et al. (2010). "A VLSI implementation of the adaptive exponential integrate-and-fire neuron model". In: *Advances in Neural Information Processing Systems*, pp. 1642–1650.

- Miyazaki, Hiroyuki et al. (2012). "Overview of the K computer System". In: *Fujitsu Scientific and Technical Journal* 48.3, pp. 255–265.
- Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533.
- Mnih, Volodymyr et al. (2016). "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*, pp. 1928–1937.
- Moreno-Bote, Rubén, David C Knill, and Alexandre Pouget (2011). "Bayesian sampling in visual perception". In: *Proc. Natl. Acad. Sci. USA* 108.30, pp. 12491–12496.
- Moreno-Bote, Ruben and Nestor Parga (2006). "Auto- and crosscorrelograms for the spike response of leaky integrate-and-fire neurons with slow synapses". In: *Phys. Rev. Lett.* 96, p. 028101.
- Moreno-Bote, Ruben et al. (2014). "Information-limiting correlations". In: *Nat. Neurosci.* 17, pp. 1410–1417.
- Morgan, Michael L, Gregory C DeAngelis, and Dora E Angelaki (2008). "Multisensory integration in macaque visual cortex depends on cue reliability". In: *Neuron* 59.4, pp. 662–673.
- Morrison, A., A. Aertsen, and M. Diesmann (2007a). "Spike-Timing Dependent Plasticity in Balanced Random Networks". In: *Neural Comput.* 19, pp. 1437–1467.
- Morrison, A., M. Diesmann, and W. Gerstner (2008). "Phenomenological models of synaptic plasticity based on spike-timing". In: *Biol. Cybern.* 98, pp. 459–478.
- Morrison, Abigail, Ad Aertsen, and Markus Diesmann (2007b). "Spike-timing dependent plasticity in balanced random networks". In: *Neural Comput.* In press.
- Morrison, Abigail et al. (2005). "Advancing the boundaries of high connectivity network simulation with distributed computing". In: *Neural Comput.* 17.8, pp. 1776–1801.
- Morrison, Abigail et al. (2007). "Exact subthreshold integration with continuous spike times in discrete time neural network simulations". In: *Neural Comput.* 19.1, pp. 47–79.
- Moser, Edvard I, Emilio Kropff, and May-Britt Moser (2008). "Place cells, grid cells, and the brain's spatial representation system". In: *Annu. Rev. Neurosci.* 31, pp. 69–89.
- Mountcastle, Vernon (1978). "An organizing principle for cerebral function: the unit model and the distributed system". In:
- Muller, Lorenz K and Giacomo Indiveri (2017). "Neural Sampling by Irregular Gating Inhibition of Spiking Neurons and Attractor Networks". In: *arXiv preprint arXiv:1605.06925*.
- Mumford, David (1992). "On the computational architecture of the neocortex". In: *Biological cybernetics* 66.3, pp. 241–251.
- Mundy, Andrew et al. (2015). "An efficient SpiNNaker implementation of the neural engineering framework". In: *Neural Networks (IJCNN), 2015 International Joint Conference on.* IEEE, pp. 1–8.
- Nageswaran, Jayram Moorkanikara et al. (2009). "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors". In: *Neural networks* 22.5, pp. 791–800.
- Nawrocki, Robert A, Richard M Voyles, and Sean E Shaheen (2016). "A Mini Review of Neuromorphic Architectures and Implementations". In: *IEEE Transactions on Electron Devices* 63.10, pp. 3819–3829.
- Neel, Louis (1952). "Antiferromagnetism and Ferrimagnetism". In: *Proceedings of the Physical Society. Section A* 65.11, p. 869.
- Neftci, Emre O et al. (2016). "Stochastic synapses enable efficient brain-inspired learning machines". In: *Frontiers in neuroscience* 10.

- Neltner, L. et al. (2000). "Synchrony in Heterogeneous Networks of Spiking Neurons". In: *Neural Comput.* 12.7, pp. 1607–1641.
- Nordlie, Eilen, Marc-Oliver Gewaltig, and Hans Ekkehard Plesser (2009a). "Towards reproducible descriptions of neuronal network models". In: *PLoS computational biology* 5.8, e1000456.
- (2009b). "Towards Reproducible Descriptions of Neuronal Network Models". In: *PLoS Comput. Biol.* 5.8, e1000456.
- O'Connor, Daniel H. et al. (2010). "Neural Activity in Barrel Cortex Underlying Vibrissa-Based Object Localization in Mice". In: *Neuron* 67.6, pp. 1048–1061.
- O'Leary, Timothy et al. (2014). "Cell Types, Network Homeostasis, and Pathological Compensation from a Biologically Plausible Ion Channel Expression Model". In: *Neuron* 82.4, pp. 809–821.
- Olshausen, Bruno A and David J Field (1997). "Sparse coding with an overcomplete basis set: A strategy employed by V1?" In: *Vision research* 37.23, pp. 3311–3325.
- Orbán, Gergő et al. (2016). "Neural variability and sampling-based probabilistic representations in the visual cortex". In: *Neuron* 92.2, pp. 530–543.
- Osborne, Leslie C et al. (2008). "The neural basis for combinatorial coding in a cortical population response". In: *J. Neurosci.* 28.50, pp. 13522–13531.
- Padmanabhan, Krishnan and Nathaniel N Urban (2010). "Intrinsic biophysical diversity decorrelates neuronal firing while increasing information content". In: *Nat. Neurosci.* 13.10, pp. 1276–1282.
- Pantazi, Angeliki et al. (2016). "All-memristive neuromorphic computing with level-tuned neurons". In: *Nanotechnology* 27.35, p. 355205.
- Parisi, Giorgio (1986). "Asymmetric neural networks and the process of learning". In: *Journal of Physics A: Mathematical and General* 19.11, p. L675.
- Pearson, Karl (1901). "LIII. On lines and planes of closest fit to systems of points in space". In: *Philosophical Magazine Series* 6 2.11, pp. 559–572. DOI: 10.1080/14786440109462720.
- Pecevski, D, L Buesing, and W Maass (2011). "Probabilistic Inference in General Graphical Models through Sampling in Stochastic Networks of Spiking Neurons". In: *PLoS Comput. Biol.* 7.12, e1002294. DOI: 10.1371/journal.pcbi.1002294.
- Perin, Rodrigo, Thomas K. Berger, and Henry Markram (2011a). "A synaptic organizing principle for cortical neuronal groups". In: *Proc. Natl. Acad. Sci. USA* 108.13, pp. 5419–5424.
- Perin, Rodrigo, Thomas K Berger, and Henry Markram (2011b). "A synaptic organizing principle for cortical neuronal groups". In: *Proc. Natl. Acad. Sci. USA* 108.13, pp. 5419–5424.
- Petersen, Carl CH and Sylvain Crochet (2013). "Synaptic computation and sensory processing in neocortical layer 2/3". In: *Neuron* 78.1, pp. 28–48.
- Petrovici, Mihai A. et al. (2014a). "Characterization and Compensation of Network-Level Anomalies in Mixed-Signal Neuromorphic Modeling Platforms". In: *PLoS ONE* 9.10, e108590. DOI: 10.1371/journal.pone.0108590.
- Petrovici, Mihai A et al. (2014b). "Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms". In: *PloS one* 9.10, e108590.
- Petrovici, Mihai A et al. (2016). "Stochastic inference with spiking neurons in the high-conductance state". In: *Phys. Rev. E* 94.4, p. 042312.
- Petrovici, Mihai A et al. (2017). "Pattern representation and recognition with accelerated analog neuromorphic systems". In: *arXiv preprint arXiv:1703.06043*.

- Pfeil, Thomas et al. (2012). "Is a 4-bit synaptic weight resolution enough?—constraints on enabling spike-timing dependent plasticity in neuromorphic hardware". In: *Frontiers in neuroscience* 6.
- Pfeil, Thomas et al. (2013a). "Neuromorphic learning towards nano second precision". In: *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE Press, pp. 1–5. DOI: 10.1109/IJCNN.2013.6706828.
- Pfeil, Thomas et al. (2013b). "Six networks on a universal neuromorphic computing substrate". In: *Front. Neurosci.* 7.11.
- Pfeil, Thomas et al. (2013c). "Six networks on a universal neuromorphic computing substrate". In: *Frontiers in neuroscience* 7.
- Pfeil, Thomas et al. (2016). "Effect of Heterogeneity on Decorrelation Mechanisms in Spiking Neural Networks: A Neuromorphic-Hardware Study". In: *Phys. Rev. X* 6.2. 021023.
- Pinker, Steven (1999). "How the mind works". In: *Annals of the New York Academy of Sciences* 882.1, pp. 119–127.
- Pitkow, Xaq, Yashar Ahmadian, and Ken D Miller (2011). "Learning unbelievable probabilities". In: *Advances in Neural Information Processing Systems*, pp. 738–746.
- Plesser, H. E. et al. (2007). "Efficient Parallel Simulation of Large-Scale Neuronal Networks on Clusters of Multiprocessor Computers". In: *Euro-Par 2007: Parallel Processing*. Ed. by A.-M. Kermarrec, L. Bougé, and T. Priol. Vol. 4641. Lecture Notes in Computer Science. Berlin: Springer-Verlag, pp. 672–681. URL: <http://www.springerlink.com/content/rw27162113p71685/?p=4833bc44d6c9442caae5202ec82d4911&pi=70>.
- Plotnikov, Dimitri et al. (2016a). "NESTML: a modeling language for spiking neurons". In: *arXiv preprint arXiv:1606.02882*.
- Plotnikov, Dimitri et al. (2016b). "NESTML: a modeling language for spiking neurons". In: *Modellierung 2016 Conference*. Vol. 254. LNI. Bonn: Bonner Köllen Verlag, pp. 93–108. URL: <http://www.se-rwth.de/publications/NESTML-a-modeling-language-for-spiking-neurons.pdf>.
- Popper, Karl (1935). *Logik der Forschung*. Wien: Springer-Verlag Wien GmbH.
- Posch, C. et al. (2014). "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output". In: *Proc. IEEE* 102.10, pp. 1470–1484. DOI: 10.1109/JPROC.2014.2346153.
- Potjans, Tobias C and Markus Diesmann (2012). "The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model". In: *Cerebral cortex* 24.3, pp. 785–806.
- Potjans, Tobias C. and Markus Diesmann (2014). "The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model". In: *Cereb. Cortex* 24.3, pp. 785–806. DOI: doi : 10.1093/cercor/bhs358.
- Potjans, W., A. Morrison, and M. Diesmann (2009). "A spiking neural network model of an actor-critic learning agent". In: *Neural Comput.* 21, pp. 301–339.
- Potjans, Wiebke, Markus Diesmann, and Abigail Morrison (2011). "An imperfect dopaminergic error signal can drive temporal-difference learning". In: *PLoS Comput. Biol.* 7.5, e1001133.
- Potjans, Wiebke, Abigail Morrison, and Markus Diesmann (2010). "Enabling functional neural circuit simulations with distributed computing of neuromodulated plasticity". In: *Front. Comput. Neurosci.* 4.141. DOI: 10.3389/fncom.2010.00141.
- Pouget, Alexandre et al. (2013). "Probabilistic brains: knowns and unknowns". In: *Nature neuroscience* 16.9, pp. 1170–1178.

- Press, William H. et al. (2007). *Numerical Recipes: The Art of Scientific Computing*. 3rd. Cambridge University Press. ISBN: 0-521-88068-8.
- Prinz, Astrid A, Dirk Bucher, and Eve Marder (2004). "Similar network activity from disparate circuit parameters". In: *Nat. Neurosci.* 7, pp. 1345–1352. URL: <http://dx.doi.org/10.1038/nn1352>.
- Probst, Dimitri et al. (2015). "Probabilistic inference in discrete spaces can be implemented into networks of LIF neurons". In: *Frontiers in computational neuroscience* 9.
- Quigley, Morgan et al. (2009). "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, p. 5.
- Raina, Rajat, Anand Madhavan, and Andrew Y Ng (2009). "Large-scale deep unsupervised learning using graphics processors". In: *Proceedings of the 26th annual international conference on machine learning*. ACM, pp. 873–880.
- Ramsey, Frank Plumpton (1950). *The Foundations of Mathematics: and other Logical Essays*. International library of psychology, philosophy and scientific method. Ed. by R. B. Braithwaite, first publ. 1931. London: Routledge & Kegan Paul.
- Rao, Rajesh PN and Dana H Ballard (1999). "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects." In: *Nature neuroscience* 2.1.
- Rasmussen, Daniel and Chris Eliasmith (2014). "A neural model of hierarchical reinforcement learning." In: *CogSci*.
- Renart, Alfonso et al. (2010a). "The asynchronous State in Cortical Circuits". In: *Science* 327, pp. 587–590.
- Renart, Alfonso et al. (2010b). "The asynchronous state in cortical circuits". In: *science* 327.5965, pp. 587–590.
- Renaud, S. et al. (2010). "PAX: A mixed hardware/software simulation platform for spiking neural networks". In: *Neural Networks* 23.7, pp. 905–916.
- Ribrault, Claire, Ken Sekimoto, and Antoine Triller (2011a). "From the stochasticity of molecular processes to the variability of synaptic transmission". In: *Nat. Rev. Neurosci.* 12.7, p. 375.
- (2011b). "From the stochasticity of molecular processes to the variability of synaptic transmission". In: *natrevnsnci* 12, pp. 375–387.
- Ringach, Dario L (2009). "Spontaneous and driven cortical activity: implications for computation". In: *Current opinion in neurobiology* 19.4, pp. 439–444.
- Robert, Christian and George Casella (2005). *Monte Carlo Statistical Methods*. 2nd ed. Springer Texts in Statistics. Springer.
- Rombouts, Jaldert O, Sander M Bohte, and Pieter R Roelfsema (2015). "How attention can create synaptic tags for the learning of working memories in sequential tasks". In: *PLoS Comput Biol* 11.3, e1004060.
- Rosenbaum, Robert and Kresimir Josic (2011). "Mechanisms That Modulate the Transfer of Spiking Correlations". In: *Neural Comput.* 23.5, pp. 1261–1305.
- Rosenbaum, Robert, Jonathan Rubin, and Brent Doiron (2012). "Short term synaptic depression imposes a frequency dependent filter on synaptic information transfer". In: *PLoS Comput. Biol.* 8.6, e1002557.
- Rosenbaum, Robert, Jonathan E Rubin, and Brent Doiron (2013). "Short-term synaptic depression and stochastic vesicle dynamics reduce and shape neuronal correlations". In: *Journal of neurophysiology* 109.2, pp. 475–484.
- Rosenblatt, Frank (1958). "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

- Rosenzweig, Michael L. (1978). "Competitive speciation". In: *Biological Journal of the Linnean Society* 10.3, pp. 275–289. DOI: 10.1111/j.1095-8312.1978.tb00016.x.
- Roth, Gerhard and Ursula Dicke (2005). "Evolution of the brain and intelligence". In: *Trends in cognitive sciences* 9.5, pp. 250–257.
- Roxin, Alex (2011). "The role of degree distribution in shaping the dynamics in networks of sparsely connected spiking neurons". In: *Front. Comput. Neurosci.* 5.8. DOI: 10.3389/fncom.2011.00008.
- Roxin, Alex et al. (2011). "On the distribution of firing rates in networks of cortical neurons". In: *J. Neurosci.* 31.45, pp. 16217–16226.
- Rueckert, Elmar et al. (2016). "Recurrent spiking networks solve planning tasks". In: *Scientific reports* 6.
- Rumelhart David, E., E. Hinton Geoffrey, and J. Williams Ronald (1986). "Learning representations by back-propagating errors". In: *Nature* 323, pp. 533–536.
- Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.
- Sakata, Shuzo and Kenneth D. Harris (2009). "Laminar Structure of Spontaneous and Sensory-Evoked Population Activity in Auditory Cortex". In: *Neuron* 64, pp. 404–418.
- Salakhutdinov, Ruslan and Geoffrey E Hinton (2009). "Deep Boltzmann Machines." In: *AISTATS*. Vol. 1, p. 3.
- Salinas, Emilio and Terrence J. Sejnowski (2001). "Correlated neuronal activity and the flow of neural information". In: *Nat. Rev. Neurosci.* 2.8, pp. 539–550.
- Samsonovich, Alexei and Bruce L McNaughton (1997). "Path integration and cognitive mapping in a continuous attractor neural network model". In: *J. Neurosci.* 17.15, pp. 5900–5920.
- Sanborn, Adam N and Nick Chater (2016). "Bayesian brains without probabilities". In: *Trends in cognitive sciences* 20.12, pp. 883–893.
- Sanz Leon, Paula et al. (2013). "The Virtual Brain: a simulator of primate brain network dynamics". In: *Front. Neuroinform.* 7, p. 10. DOI: 10.3389/fninf.2013.00010. URL: <http://journal.frontiersin.org/article/10.3389/fninf.2013.00010>.
- Savin, Cristina and Sophie Deneve (2014). "Spatio-temporal representations of uncertainty in spiking neural networks". In: *Advances in Neural Information Processing Systems*, pp. 2024–2032.
- Schemmel, J. et al. (2006). "Implementing synaptic plasticity in a VLSI spiking neural network model". In: *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN)*. Vancouver: IEEE Press, pp. 1–6.
- Schemmel, Johannes et al. (2010a). "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling". In: *Proceedings of the 2010 International Symposium on Circuits and Systems (ISCAS)*. Paris: IEEE Press, pp. 1947–1950.
- Schemmel, Johannes et al. (2010b). "A wafer-scale neuromorphic hardware system for large-scale neural modeling". In: *Circuits and systems (ISCAS), proceedings of 2010 IEEE international symposium on*. IEEE, pp. 1947–1950.
- Schenck, Wolfram, Yury V. Zaytsev, and Abigail Morrison (2014). "Performance model for large-scale neural simulations with NEST". In: *Extended Poster Abstracts of the SC14 Conference for Supercomputing*. New Orleans, LA.
- Schenck, Wolfram et al. (2014). "Performance Model for Large-Scale Neural Simulations with NEST". In: *Extended Poster Abstracts of the SC14 Conference for Supercomputing*. New Orleans (LA).

- Schlaifer, Robert and Howard Raiffa (1968). *Applied statistical decision theory*. Division of Research, Harvard Business School, Boston.
- Schmidt, Maximilian et al. (2016). "Full-density multi-scale account of structure and dynamics of macaque visual cortex". In: *arXiv preprint arXiv:1511.09364v4*.
- Schmitt, Sebastian et al. (2017). "Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system". In: *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, pp. 2227–2234.
- Schmuker, Michael, Thomas Pfeil, and Martin Paul Nawrot (2014). "A neuromorphic network for generic multivariate data classification". In: *Proc. Natl. Acad. Sci. USA* 111.6, pp. 2081–2086. DOI: 10.1073/pnas.1303053111.
- Schmuker, Michael and Gisbert Schneider (2007). "Processing and classification of chemical data inspired by insect olfaction". In: *Proc. Natl. Acad. Sci. USA* 104.51, pp. 20285–20289. DOI: 10.1073/pnas.0705683104.
- Schuecker, Jannis et al. (2017). "Fundamental Activity Constraints Lead to Specific Interpretations of the Connectome". In: *PLoS Comput. Biol.* 13.2, pp. 1–25. DOI: 10.1371/journal.pcbi.1005179. URL: <http://dx.doi.org/10.1371/journal.pcbi.1005179>.
- Schuman, Catherine D et al. (2017). "A Survey of Neuromorphic Computing and Neural Networks in Hardware". In: *arXiv preprint arXiv:1705.06963*.
- Schwartz, Eric L. (1990). *Computational neuroscience*. The MIT Press.
- Schwartz, Marc-Olivier (2013). "Reproducing Biologically Realistic Regimes on a Highly-Accelerated Neuromorphic Hardware System". PhD thesis. Ruprecht-Karls-Universität Heidelberg.
- Sedgewick, Robert and Kevin Wayne (2011). *Algorithms*. 4th ed. Addison-Wesley Professional.
- Sejnowski, Terrence J, Patricia S Churchland, and J Anthony Movshon (2014). "Putting big data to good use in neuroscience". In: *Nature neuroscience* 17.11, pp. 1440–1441.
- Sengupta, Abhronil et al. (2016). "Magnetic tunnel junction mimics stochastic cortical spiking neurons". In: *Scientific reports* 6, p. 30039.
- Senk, Johanna et al. (2015). "From randomly connected to spatially organized multi-layered cortical network models". In: *11th Goettingen Meeting of the German Neuroscience Society*.
- SenseMaker (2009). *SenseMaker: A Multi-sensory, Task-specific, Adaptable perception System; project website*. Available at: http://cordis.europa.eu/project/rcn/62746_en.html.
- Serb, Alexander et al. (2016). "Unsupervised learning in probabilistic neural networks with multi-state metal-oxide memristive synapses". In: *Nature communications* 7, p. 12611.
- Serrano-Gotarredona, R. et al. (2009). "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory - Processing - Learning - Actuating System for High-Speed Visual Object Recognition and Tracking". In: *IEEE Trans. Neural Netw.* 20.9, pp. 1417–1438. DOI: 10.1109/TNN.2009.2023653.
- Shadlen, Michael N. and William T. Newsome (1998). "The Variable Discharge of Cortical Neurons: Implications for Connectivity, Computation, and Information Coding". In: *J. Neurosci.* 18.10, pp. 3870–3896.
- Shafi, M. et al. (2007). "Variability in neuronal activity in primate cortex during working memory tasks". In: *Neuroscience* 146.3, pp. 1082–1108.
- Shamir, Maoz and Haim Sompolinsky (2006). "Implications of neuronal diversity on population coding". In: *Neural Comput.* 18.8, pp. 1951–1986.

- Sharp, Thomas et al. (2012). "Power-efficient simulation of detailed cortical micro-circuits on SpiNNaker". In: *J. Neurosci. Methods* 210.1, pp. 110–118.
- Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.
- Softky, William R. and Christof Koch (1993). "The Highly Irregular Firing of Cortical Cells Is Inconsistent with Temporal Integration of Random EPSPs". In: *J. Neurosci.* 13.1, pp. 334–350.
- Song, Sen et al. (2005). "Highly nonrandom features of synaptic connectivity in local cortical circuits". In: *PLoS Biol.* 3.3, e68.
- Spletstoesser, Thomas (2017). *SciStyle*. Licensed under the Creative Commons Attribution-Share Alike 4.0 International license. URL: <http://www.scistyle.com/>.
- Stein, R. B., E. R. Gossen, and K. E. Jones (2005). "Neuronal Variability: Noise or Part of the Signal?" In: *natrevnsci* 6, pp. 389–397.
- Stepanyants, Armen et al. (2009). "The fractions of short- and long-range connections in the visual cortex". In: *Proc. Natl. Acad. Sci. USA* 106.9, pp. 3555–3560.
- Stevenson, Ian H and Konrad P Kording (2011). "How advances in neural recording affect data analysis". In: *Nature neuroscience* 14.2, pp. 139–142.
- Stocks, Nigel G. (2000). "Suprathreshold stochastic resonance in multilevel threshold systems". In: *Physical Review Letters* 84.11, p. 2310.
- Stromatias, Evangelos et al. (2015a). "Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in neuroscience* 9.
- Stromatias, Evangelos et al. (2015b). "Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on spinnaker". In: *Neural Networks (IJCNN), 2015 International Joint Conference on.* IEEE, pp. 1–8.
- Sundareswara, Rashmi and Paul R Schrater (2008). "Perceptual multistability predicted by search model for Bayesian decisions". In: *Journal of vision* 8.5, pp. 12–12.
- Sussillo, David and Larry F Abbott (2009a). "Generating coherent patterns of activity from chaotic neural networks". In: *Neuron* 63.4, pp. 544–557.
- (2009b). "Generating coherent patterns of activity from chaotic neural networks". In: *Neuron* 63.4, pp. 544–557.
- Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press.
- Swadlow, Harvey A. (1988). "Efferent Neurons and Suspected Interneurons in Binocular Visual Cortex of the Awake Rabbit: Receptive Fields and Binocular Properties". In: *J. Neurophysiol.* 59.4, pp. 1162–1187.
- Tabareau, Nicolas, Jean-Jacques Slotine, and Quang-Cuong Pham (2010). "How Synchronization Protects from Noise". In: *PLoS Comput. Biol.* 6.1. doi:10.1371/journal.pcbi.1000637.
- Téglás, Ernő et al. (2011). "Pure reasoning in 12-month-old infants as probabilistic inference". In: *science* 332.6033, pp. 1054–1059.
- Tetzlaff, T. et al. (2004). "Consequences of realistic network size on the stability of embedded synfire chains". In: *Neurocomputing* 58–60, pp. 117–121.
- Tetzlaff, Tom et al. (2008). "Dependence of neuronal correlations on filter characteristics and marginal spike-train statistics". In: *Neural Comput.* 20.9, 2133–2184.
- Tetzlaff, Tom et al. (2012a). "Decorrelation of neural-network activity by inhibitory feedback". In: *PLoS Comput. Biol.* 8.8, e1002596.
- Tetzlaff, Tom et al. (2012b). "Decorrelation of neural-network activity by inhibitory feedback". In: *PLoS Comput Biol* 8.8, e1002596.

- Thakur, Rajeev et al. (2010). "MPI at Exascale". In: *Proceedings of SciDAC 2*, pp. 14–35.
- Thomson, Eric E, Rafael Carra, and Miguel AL Nicolelis (2013). "Perceiving invisible light through a somatosensory cortical prosthesis". In: *Nature communications* 4, p. 1482.
- Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "Mujoco: A physics engine for model-based control". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, pp. 5026–5033.
- Trichina, E. et al. (2001). "Supplemental cryptographic hardware for smart cards". In: *Micro, IEEE* 21.6, pp. 26–35. ISSN: 0272-1732. DOI: 10.1109/40.977755.
- Tripathy, Shreejoy J. et al. (2013). "Intermediate intrinsic diversity enhances neural population coding". In: *Proc. Natl. Acad. Sci. USA* 110.20, pp. 8248–8253. DOI: 10.1073/pnas.1221214110. eprint: <http://www.pnas.org/content/110/20/8248.full.pdf+html>. URL: <http://www.pnas.org/content/110/20/8248.abstract>.
- Tripp, Bryan and Chris Eliasmith (2007). "Neural populations can induce reliable postsynaptic currents without observable spike rate changes or precise spike timing". In: *Cereb. Cortex* 17.8, pp. 1830–1840.
- Tsodyks, M., I. Mitkov, and H. Sompolinsky (1993). "Pattern of synchrony in inhomogeneous networks of oscillators with pulse interactions". In: *Phys. Rev. Lett.* 71.8.
- Tuma, Tomas et al. (2016). "Stochastic phase-change neurons". In: *Nature nanotechnology* 11.8, pp. 693–699.
- Turing, Alan Mathison (1936). "On computable numbers, with an application to the Entscheidungsproblem". In: *Proceedings of the London mathematical society* 42, pp. 230–265.
- Tversky, Amos and Daniel Kahneman (1973). "Availability: A heuristic for judging frequency and probability". In: *Cognitive psychology* 5.2, pp. 207–232.
- (1985). "The framing of decisions and the psychology of choice". In: *Environmental Impact assessment, technology assessment, and risk analysis*. Springer, pp. 107–129.
- (1992). "Advances in prospect theory: Cumulative representation of uncertainty". In: *Journal of Risk and uncertainty* 5.4, pp. 297–323.
- Urbanczik, Robert and Walter Senn (2009). "Reinforcement learning in populations of spiking neurons". In: *Nature neuroscience* 12.3, p. 250.
- (2014). "Learning by the dendritic prediction of somatic spiking". In: *Neuron* 81.3, pp. 521–528.
- Van Albada, Sacha J et al. (2016). "Full-scale simulation of a cortical microcircuit on SpiNNaker". In: *Frontiers in Neuroinformatics Conference Abstract: Neuroinformatics*.
- Van Kampen, Nicolaas Godfried (1992). *Stochastic processes in physics and chemistry*. Vol. 1. Elsevier.
- Van Vreeswijk, C. and H. Sompolinsky (1998). "Chaotic balanced state in a model of cortical circuits". In: *Neural Comput.* 10.6, pp. 1321–1371.
- Vanarse, Anup, Adam Osseiran, and Alexander Rassau (2016). "A review of current neuromorphic approaches for vision, auditory, and olfactory sensors". In: *Frontiers in neuroscience* 10.
- Vasilaki, E et al. (2009). "Spike-Based Reinforcement Learning in Continuous State and Action Space: When Policy Gradient Methods Fail". In: *PLoS Comput. Biol.* 5.12, e1000586. doi:10.1371/journal.pcbi.1000586.

- Vogelstein, R.J. et al. (2007). "Dynamically Reconfigurable Silicon Array of Spiking Neurons With Conductance-Based Synapses". In: *IEEE Trans. Neural Netw.* 18.1, pp. 253–265.
- Von Helmholtz, Hermann (1867). *Handbuch der physiologischen Optik*. Vol. 9. Voss.
- Von Neumann, John (1945). *First Draft of a Report on the EDVAC*. Tech. rep. University of Pennsylvania.
- Vreeswijk, Carl van and Haim Sompolinsky (1996a). "Chaos in Neuronal Networks with Balanced Excitatory and Inhibitory Activity". In: *Science* 274, pp. 1724–1726.
- (1996b). "Chaos in Neuronal Networks with Balanced Excitatory and Inhibitory Activity". In: *Science* 274, pp. 1724–1726.
- (1998). "Chaotic balanced state in a model of cortical circuits". In: *Neural computation* 10.6, pp. 1321–1371.
- Vul, Edward and Harold Pashler (2008). "Measuring the crowd within: Probabilistic representations within individuals". In: *Psychological Science* 19.7, pp. 645–647.
- Wald, Abraham (1939). "Contributions to the theory of statistical estimation and testing hypotheses". In: *The Annals of Mathematical Statistics* 10.4, pp. 299–326.
- Watkins, Christopher John Cornish Hellaby (1989). "Learning from delayed rewards". PhD thesis. University of Cambridge England.
- Weidel, Philipp et al. (2016). "Closed loop interactions between spiking neural network and robotic simulators based on MUSIC and ROS". In: *Frontiers in neuroinformatics* 10.
- Welling, Max and Yee Whye Teh (2003). "Approximate inference in Boltzmann machines". In: *Artificial Intelligence* 143.1, pp. 19–50.
- Wellman, Michael P and Max Henrion (1993). "Explaining'explaining away'". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.3, pp. 287–292.
- Wernicke, Carl (1906). "The symptom complex of aphasia". In: *Diseases of the nervous system*. Ed. by A. Church. New York: Appleton, pp. 265–324.
- White, John A, Jay T Rubinstein, and Alan R Kay (2000). "Channel noise in neurons". In: *Trends in neurosciences* 23.3, pp. 131–137.
- Wiechert, Martin T et al. (2010). "Mechanisms of pattern decorrelation by recurrent neuronal circuits". In: *Nat. Neurosci.* 13, pp. 1003–1010.
- Williams, Stephen R and Simon J Mitchell (2008). "Direct measurement of somatic voltage clamp errors in central neurons". In: *Nature neuroscience* 11.7, pp. 790–798.
- Wils, Stefan and Erik De Schutter (2009). "STEPS: modeling and simulating complex reaction-diffusion systems with Python". In: *Front. Neuroinformatics* 3.15.
- Yang, Minhao et al. (2016). "A 0.5V 55 μ W 64 $\times$ 2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing". In: *IEEE Journal of Solid-State Circuits* 51.11, pp. 2554–2569.
- Yim, Man Yi, Ad Aertsen, and Stefan Rotter (2013). "Impact of intrinsic biophysical diversity on the activity of spiking neurons". In: *Phys. Rev. E* 87.
- Yu, T. and G. Cauwenberghs (2010). "Analog VLSI Biophysical Neurons and Synapses With Programmable Membrane Channel Kinetics". In: *IEEE Trans. Biomed. Circuits Syst.* 4.3, pp. 139–148. DOI: 10.1109/TBCAS.2010.2048566.
- Zängl, Günther et al. (2015). "The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core". In: *Quarterly Journal of the Royal Meteorological Society* 141.687, pp. 563–579.
- Zhang, Calvin and Charles S Peskin (2015). "Improved signaling as a result of randomness in synaptic vesicle release". In: *Proc. Natl. Acad. Sci. USA* 112.48, pp. 14954–14959.

Author's list of publications

The work presented in this thesis is in parts based on the following publications:

- Jakob Jordan, Mihai A. Petrovici, Oliver Breiwwieser, Johannes Schemmel, Karlheinz Meier, Markus Diesmann, and Tom Tetzlaff (2017b). "Stochastic neural computation without noise". In: *ArXiv e-prints*. arXiv: 1710.04931 [q-bio.NC]
Results presented in chapter 5.
- Thomas Pfeil, Jakob Jordan, Tom Tetzlaff, Andreas Grübl, Johannes Schemmel, Markus Diesmann, and Karlheinz Meier (2016). "Effect of Heterogeneity on Decorrelation Mechanisms in Spiking Neural Networks: A Neuromorphic-Hardware Study". In: *Phys. Rev. X* 6.2. 021023
Results presented in chapter 6.
- Jakob Jordan, Tammo Ippen, Moritz Helias, Itaru Kitayama, Mitsuhsa Sato, Jun Igarashi, Markus Diesmann, and Susanne Kunkel (2017a). "Extremely scalable spiking neural network simulation code: from laptops to exascale computers". (submitted)
Results presented in chapter 8.
- Jakob Jordan, Philipp Weidel, and Abigail Morrison (2017). "Closing the loop between neural network simulators and the OpenAI Gym". In: *ArXiv e-prints*. arXiv: 1709.05650 [q-bio.NC]
Results presented in chapter 9.

Author contributions are indicated in the beginning of the respective chapters.

Acknowledgements

This thesis would not have been possible without the support of many people, guiding me, providing input and keeping me company during these four years.

First of all I would like to thank Markus Diesmann for giving me the opportunity to pursue my PhD at his institute, for never tiring of teaching scientific writing and showing me how to handle the diplomatic aspects of science. Also for providing an excellent scientific environment which gives me the freedom to pursue my own interests while at the same time fosters tight interaction with colleagues.

Second I would like to thank Professor Leibe for agreeing so enthusiastically to acting as a second reviewer on this thesis.

I am deeply indebted to my direct supervisor Tom Tetzlaff for teaching me how to science! Our endless discussions, his broad knowledge, strong work ethic, thoroughness and fear of color have significantly impacted my development as a scientist. He is a true role model.

Of course I would like to thank Sonja Grün for our discussions on the role of noise in the brain. There are no useless spikes!

I would like to extend my thanks to Susanne Kunkel and Moritz Helias for always providing incredibly helpful input from their enormous pool of experience.

Many thanks go to Jannis Schuecker for the adventurous first years (“the dish washer broke!”) and for never getting tired of questioning every single statement of mine. This led to many, many heated discussions on all aspects of life and science that also significantly shaped ideas expressed in this thesis. The same extends to Maximilian Schmidt, who also questions everything, at the same time seems to know everything and who is a formidable kicker opponent. This brings me to Pietro Quaglio who I would like to thank for his tireless effort of keeping human hubris in check and for defending sweaters no matter who they belong to. I would like to thank Philipp Weidel for providing a solid balance to my optimism, for keeping it real and, together with Abigail Morrison, for our short, but super productive collaboration. Without Alper Yegenouglu, my workflow would be much less efficient and definitely have much less bling-bling. Thank you to Jochen Eppler, David Dahmen, Hannah Bos, Jonas Stappmans, for the pleasant office times, and especially to Johanna Senk for keeping me company over the years, also during extended evening sessions.

Many thanks to PierGianLuca Porta Mana for showing me the path to “the one and only True Religion”, for being a tremendous source of inspiration and motivation, and our science breakfasts. Without these, my thinking would be much more clouded.

Thank you of course to everyone from Heidelberg, Johannes Schemmel, Andreas Grübl and Karlheinz Meier and everyone else I had the pleasure of meeting over the years. Big props especially to Mihai Petrovici, Oliver Breitwieser and Thomas Pfeil for the intense collaboration and exciting extracurricular activities.

I would like to thank Bernd Wiebelt and Michael Bontenackels for their help with various software and hardware related issues and for keeping our local cluster in shape. The same goes for Tiziano Zito who I would additionally like to thank

those great, pythonic summer schools. Thank you also to Zbyszek Szmek, Etienne Roesch and everyone else from #ASPP for the enlightening and fun times!

Without the help of our administrative staff, our institute would not be running as smoothly. Thank you for your work, especially Janine Lehm for always being very patient with me.

Special thanks go to Julia Sprenger for keeping the blood sugar of the institute high and making the Python Module of the Week an actual thing. Keep up the good work!

To all current and former members of INM-6 and Ruebenhausen not mentioned here personally due to time and sanity constraints: Thank you for contributing to this exciting research environment and the stimulating discussions. Of course special thanks go to the Kicker crew for the intense physical competition that provides a great balance to the brain-heavy parts of our daily routine.

This work would not have been possible without the tireless efforts of the scientific Python, NEST, Emacs and L^AT_EX communities. Thank you for all your hard work!

Vielen Dank an meine Eltern und meine Oma die mich stetig unterstützt und immer wieder ermutigt haben. Ohne euch wäre diese Arbeit nie möglich gewesen.

Ein besonderer Dank gebührt Maria, die mich in den letzten Monaten (Jahren!) stets aufopferungsvoll und mit unendlicher Geduld begleitet hat. Danke dafür!

Funding

This research was partly supported by the Helmholtz Alliance through the Helmholtz Association portfolio theme Supercomputing and Modeling for the Human Brain (SMHB), the Initiative and Networking Fund of the Helmholtz Association, the Helmholtz young investigator group VH-NG-1028, the Jülich Aachen Research Alliance (JARA), EU Grant No. 269921 (BrainScaleS), EU Grant No. 604102 (Human Brain Project, HBP), EU Grant No. 720270 (HBP SGA1), the Deutsche Forschungsgemeinschaft and Ruprecht-Karls-Universität Heidelberg through the funding programme Open Access Publishing, the German Federal Ministry of Education through the German-Japanese Computational Neuroscience Project (BMBF Grant 01GQ1343), and EuroSPIN.

Use of the JUQUEEN supercomputer in Jülich was made possible by VSR computation time grant JINB33. This research used resources of K computer at the RIKEN Advanced Institute for Computational Science. Supported by the project Exploratory Challenge on Post-K Computer (Understanding the neural mechanisms of thoughts and its applications to AI) of the Ministry of Education, Culture, Sports, Science and Technology (MEXT).