The following full text is a publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/199748

Please be advised that this information was generated on 2019-09-05 and may be subject to change.
SELF-ORGANIZATION, LEARNING & CONTROL:
PERSPECTIVES ON COMPLEX SYSTEMS.

Proefschrift ter verkrijging van de graad van doctor aan de Radboud Universiteit Nijmegen op gezag van de rector magnificus prof. dr. J.H.J.M. van Krieken, volgens besluit van het college van decanen in het openbaar te verdedigen op

maandag 21 januari 2019,

om 10.30 uur precies

door

Dominik Johannes Bartholomäus Thalmeier

geboren op 17 augustus 1985

te Wasserburg am Inn, Duitsland
Promotor: prof. dr. H.J. Kappen
Copromotor: dr. V. Gómez (Universitat Pompeu Fabra, Spanje)

Manuscriptcommissie:
prof. dr. P.H.E. Tiesinga
dr. W.A.J.J. Wiegerinck
prof. dr. G. Neumann (University of Lincoln, VK)
Acknowledgement

Het was een leuke en belangrijke tijd voor mij in Nederland. Ik heb veel kunnen ervaren en leren over wetenschappelijke belangen, maar ook over het leven. En, niet te vergeten, een nieuwe taal.

Bert especially you I thank, for the chance to work in the Netherlands, for the freedom I had that allowed me to explore my own scientific questions, which also means —and I think this is especially important to become a critical mind— the freedom to make mistakes. Freedom, but also criticism and many, many discussions which often gave me the necessary clarity that I needed. You helped me to ground my thinking in the details, while I was often flying high in wild abstractions. I believe that you helped me sharpen my way of thinking, I became more critical to my own ideas and reasoning.

Raoul thank you a lot for guiding me through a significant part of my PhD. I loved the discussions we had, they were an intellectual pleasure. Vicenc, it was a real pleasure to work with you. I experienced the collaboration we had as efficient and professional, which I think is not easy, as most of it was done via skype. I learned a lot from you!

Giel thank you for being a critical and interested discussion partner. You helped me to get a grip on many ideas and questions that chaotically floated through my mind! Hans, we shared many ups and downs during our time in Nijmegen. One of the ups for me was when we partnered in programming, I enjoyed that very much. This is how work life should be! Nils, Jakob, Katharina, Robert, Felix and Regina I am deeply thankful too you! I thank you for your company and many discussions that were able to spark my motivation in times when I needed it. Nils I that you for your constructive feedback on my work! Baha, so often I had questions that you helped me with. Thank you! Verena, thank you for the design of the thesis cover, it is marvellous! Guenther and the postmasters at the CNCZ thank you for your help and patience. Annet, you are amazing, thank you for your help in countless things. Sep I thank you for your help and company! Jacob, Erwin and Wim I thank you for great collaborations on the projects we worked on together. I enjoyed it a lot. I also want to thank Larry Abbot for a great discussion we had about my work on spiking neurons.
Acknowledgement

Floor you brought light and shadow to my time in the Netherlands. The one I enjoyed, from the other I learned. Thank you! Silvana I thank you for helping me through hard times I had at the end of my PhD!
# Contents

Acknowledgement iii

## 1 Introduction

1.1 Self-organization: How interaction networks of diffusing proteins can form spatial structures. 2
1.2 Learning: How networks of spiking neurons can learn system dynamics. 3
1.3 Control: How synthetic systems can learn control strategies.
   1.3.1 Dynamic programming and the Bellman equation 4
   1.3.2 Kullback-Leibler control problems allow to formulate control problems as inference problems 5
   1.3.3 Solve control problems with Monte Carlo methods 6
1.4 Conclusion: Principles of prediction and control 7

## 2 Geometry induced protein pattern formation

2.1 Abstract 9
2.2 Introduction 10
2.3 A generic reaction module for sensing of cell geometry 11
2.4 Results 13
   2.4.1 The impact of cell geometry on protein gradients in elongated cells 13
   2.4.2 Why geometry influences patterning 14
   2.4.3 Cytosolic reaction volume determines the pattern 15
   2.4.4 Pattern formation does not require a dynamical instability 16
2.5 Discussion 21
2.6 Computational methods and initial conditions 22
2.7 Supplement: Geometry induced protein pattern formation 23
   2.7.1 Model equations 23
   2.7.2 Orthogonal Elliptical Coordinates and Differential Operators 23
   2.7.3 Supplementary details for the Figures 25
   2.7.4 Equations of the minimal system 27
### CONTENTS

2.7.5 Balance of membrane affinities ........................................... 29  
2.7.6 Breaking the homogenous steady state in the circular limit .... 31  
2.7.7 Continuum limit of the boundary conditions of the minimal system ................................................................. 32  
2.7.8 The effect of volume exclusion by the nucleoid ................. 34  
2.7.9 Caption figure 2.8 ............................................................... 35

3 Learning universal computations with spikes 41  
3.1 Abstract ............................................................................. 41  
3.2 Introduction ....................................................................... 41  
3.3 Results ................................................................................ 43  
3.3.1 Continuous signal coding spiking neural networks (CSNs) . 43  
3.3.2 Learning universal computations ...................................... 53  
3.3.3 Applications ................................................................. 55  
3.4 Discussion ......................................................................... 65  
3.5 Methods ........................................................................... 69  
3.5.1 Network simulation .......................................................... 69  
3.5.2 Training phase ............................................................... 70  
3.5.3 Learning rules ............................................................... 70  
3.5.4 Control task .................................................................. 71  
3.5.5 Figure details ............................................................... 73  
3.6 Supplement: Learning universal computations with spikes ... 76  
3.6.1 Networks with nonlinear dendrites ................................... 76  
3.6.2 A sufficient condition for the echo state property of the dynamics Equation (3.6) ....................................................... 78  
3.6.3 Comparison of PCSNs with Poisson coding learning networks 80  
3.6.4 Robustness against noise .............................................. 83  
3.6.5 Robustness of PCSNs with nonlinear dendrites against structural perturbations .................................................. 83  
3.6.6 Comparison of network sizes and the spike rate-network size trade-off ............................................................... 88  
3.6.7 Error evolution for longer times ..................................... 88  
3.6.8 Details on the supporting figures ................................... 92

4 Action selection in growing state spaces: Control of Network Structure Growth 97  
4.1 Abstract ............................................................................. 97  
4.2 Introduction ....................................................................... 98  
4.3 Optimal Network Growth as a Control Problem ............... 99  
4.4 Approximating the network growth problem by a Kullback-Leibler control problem ............................................... 100  
4.4.1 Kullback-Leibler control ............................................... 100  
4.4.2 A Tractable Example .................................................... 102
4.4.3 Sampling from the KL-optimally controlled dynamics 104
4.4.4 Action selection using the KL-approximation 106
4.5 Application to Conversation Threads 107
   4.5.1 Structural Cost Function 109
   4.5.2 Uncontrolled Dynamics for Online Conversation Threads 109
   4.5.3 Control interaction 110
   4.5.4 Experimental Setup 110
   4.5.5 Experimental Results 111
4.6 Discussion 114
4.7 Supplement: Action selection in growing state spaces 116
   4.7.1 Adaptive Importance Sampling for KL-Optimal Control Computation using the Cross-Entropy method 116
   4.7.2 Analyzing the KL-optimal cost-to-go based action selection 119

5 Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization 121
   5.1 Abstract 121
   5.2 Introduction 122
   5.3 Path integral control problems 123
      5.3.1 Direct cost-optimization using gradient descent 124
      5.3.2 The PICE method 125
   5.4 Interpolating between the two methods: Smoothing stochastic control problems 126
      5.4.1 Computing the smoothed cost and its gradient 126
      5.4.2 PICE, direct cost-optimization and risk sensitivity as limiting cases of smoothed cost optimization 128
   5.5 The effect of cost function smoothing on policy optimization 129
   5.6 Numerical Method 131
      5.6.1 Compute smoothed and direct updates using natural gradients 131
      5.6.2 Reliable gradient estimation using adaptive smoothing 132
      5.6.3 Formulating a model free algorithm 132
   5.7 Numerical Experiments 133
      5.7.1 A simple LQ control problem: Brownian Viapoints 134
      5.7.2 Nonlinear control problems 134
   5.8 Discussion 141
   5.9 Supplement: Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization 143
      5.10 Derivation of the policy gradient 143
      5.11 Replacing minimization over \( u \) by minimization over \( p' \) 143
      5.12 Minimizer of smoothed cost 145
      5.13 Derivation of the gradient of the smoothed cost function 146
      5.14 Global maximum preserved under full parametrization 146
      5.15 Smoothed updates for small update steps \( \delta \) 147
Living and social systems are made up of interacting parts that give rise to complex phenomena. Molecules self-organize into cells that replicate, metabolize, move, process information and interact. Interacting cells form multicellular organs and organisms. As we go up another level in the hierarchy of complex systems, we find that these high level living systems interact with each other: Animals and humans communicate and form networks of social interactions. Due to the ubiquitousness of complex systems and the powerful mechanisms they exhibit understanding how to predict and control the behaviour of complex systems is an important endeavour.

The remainder of this introduction is structured by three different perspectives we take on complex systems—Self-organization, Learning and Control—followed by a conclusion.

1. **Self-organization:** How interaction networks of diffusing proteins can form spatial structures. The basis of complex systems is self-organization: complex structures emerge from interacting components. We discuss this phenomenon on the example of the most basic living system: a single cell. In particular we address the question of how a cell can divide in two equally sized parts. We explain that this is done by proteins that self-organize into a spatial pattern to provide a chemical signal at the center of the cell. In chapter 2 we introduce a protein interaction network that exhibits a novel mechanism that forms such a template-pattern by adapting to geometrical cues.
1. Introduction

2. Learning: How networks of spiking neurons can learn system dynamics. In this part we go beyond single cells and study a system of interconnected cells: neural networks. Neural networks are especially interesting multi-cellular systems, as they are able to learn models of other complex systems they are interacting with and are able to learn strategies how to manipulate these external systems. Understanding the mechanism behind these phenomena may thus not only help us to understand the system under consideration, but could also guide us in developing strategies how to build models and control complex —living, technological and may be even social— systems. In chapter 3 we address the question how a network of neurons that interact with electrical spikes can learn to memorize and mimic the dynamics of external systems. We present a combination of a neural coding scheme and a learning scheme that together enable spiking neural networks to learn complex dynamics and tasks. Further we demonstrate how this ability to learn a model of an external system enables a neural system to compute a control strategy to achieve a given goal.

3. Control: How synthetic systems can learn control strategies. In this part we discuss in more detail how a learned model can be used to compute a control strategy to achieve a certain goal. We leave the realm of biological modelling and focus on synthetic systems to formally examine the question of how a control strategy can be learned. We give a short introduction to the framework of optimal control and explain that optimal actions can be computed by iteratively drawing samples from a model of the system that should be controlled. In chapter 4 we extend a sampling based optimal control algorithm from continuous to discrete systems and show that this allows to compute a strategy to control the topology of an interaction network that grows with time. In chapter 5 we take a closer look at the principles behind these control algorithms. We introduce and study a new spectrum of control algorithms that brings a new perspective on some well-known algorithms which make up the limiting cases of this spectrum. Based on this we develop a new algorithm that is superior in performance to these known border cases.

1.1 Self-organization: How interaction networks of diffusing proteins can form spatial structures.

One of the earliest quantitative models how matter can self-organize into spatial structures was introduced by Alan Turing. The Turing mechanism is based on a reaction-diffusion system with a slowly diffusing activator molecule and a fast diffusing inhibitor molecule. A linear stability analysis shows that in this system, the homogeneous state is unstable to small disturbances with a stripe pattern of a
1.2 Learning: How networks of spiking neurons can learn system dynamics.

Higher organisms, like humans, that possess a central nervous system can memorize and mimic the dynamical pattern of temporal signals. This can for example be used to learn a movement pattern of muscles or to predict future events. Such learning is accomplished by recurrently connected networks of nerve cells in the brain that communicate with short electrical impulses called spikes. In the following we address the question how such networks may perform the learning.

A possible principle for learning in recurrent networks is the reservoir computing approach [Jae01,MNM02,JH04,MJS07,SA09]. In reservoir computing a recurrently connected network with random but fixed connections is used to project a time-varying input signal into a high dimensional neuron space. This high dimensional representation of the history of the input signal can then act as a feature space to a single readout neuron. This allows to train the readout neuron with simple learning algorithms to compute arbitrary non-linear mappings of the input sequence.

This mechanism of pattern formation by dynamical instabilities of the homogeneous state is a basic paradigm in understanding how pattern in biological systems arise. A well studied example of pattern formation is the Min-system, which regulates cell division in Escherichia Coli. The dynamics of the Min-protein reaction-network yields a pattern that provides a chemical signal where the center of the cell is located. This results in cell division proteins forming a ring in the very center of the cell, allowing the cell to be split exactly in the middle. The paradigm that pattern arise as a dynamical instability has also been used to understand pattern formation in the Min-system. A problem with this approach is that a proper adjustment of the resulting pattern to the geometry of the cell requires a fine-tuning of the reaction parameters. This is in contrast to the experimental finding that the Min-pattern in Escherichia Coli show robustness to perturbations of the cell geometry.

In chapter 2 we explore the principles behind the robust geometry adaptation of protein pattern. To this end we take a modelling approach that considers some biological details that are usually neglected: cytosolic degradation processes and a more realistic geometrical model of the cell. On the example of a minimal protein interaction we demonstrate that these biological details enable robust adaptation of the protein pattern to cell-geometry that does not require a fine tuning of the reaction parameters. An analysis of this mechanism reveals that we found a new principle of self organization in cells that does not rely on dynamic instabilities but on an emergent preference of the protein-density to certain geometrical cues.
1. Introduction

With the FORCE method [SA09], the scope of reservoir computing can be further extended to the robust learning of temporal signals. [SA09] start with a chaotic neural network and a readout neuron with random connections to the reservoir. They then connect the readout neuron to the input of the recurrent neural network and train it with a fast learning algorithm to mimic a target signal. After a while training can be stopped and the neural network can independently reproduce the temporal sequence of the target signal.

Reservoir computing with FORCE is thus a powerful method for learning versatile computations. However, as it was only demonstrated for a very abstracted model of neurons that communicate via continuous signals instead of spikes, it remained an open question whether this is a suitable model for learning in the brain (see [ADM16] for a review). We address this question in chapter 3 and demonstrate that the FORCE method can indeed also be used for networks of spiking neurons. To do this, we employ a precise spike-coding scheme [BBMD12] that allows to construct a reservoir of spiking neurons that emulates a reservoir of neurons that communicate via continuous signals. With this precise coding reservoir we can reproduce the key results of [SA09] but this time with networks of spiking neurons. In addition we present a slightly modified version of the FORCE method that allows to go beyond the learning of temporal signals and learn the entire dynamics of an external system including its response to input. We use this to learn the input-output behaviour of a simple mechanical system and demonstrate that this learned model can be used by a model based control algorithm to compute a control strategy that moves the dynamical system into a desired state.

1.3 Control: How synthetic systems can learn control strategies.

Optimal control theory is about how to compute the optimal external input to a system so that the system behaves in a desired way. For this sake a cost function is formulated that maps the sequence of control-inputs and system-states to a scalar value that is low when the behaviour of the system is desired and high when it is undesired. For deterministic systems the optimal control minimizes this cost function, for stochastic systems the expected value of the cost is minimized.

1.3.1 Dynamic programming and the Bellman equation

For the discrete time, finite horizon case\(^1\), the optimal control problem can be formally solved using dynamic programming, which breaks the multi-period planning problem into simpler steps at different points in time. To this end the optimal cost-to-go is introduced that tells for each time and state what cost is incurred in

\(^1\)For finite horizon problems the cost is defined on a finite time window.
the future, given the agent behaves optimally from this point on. To compute it, the optimal cost-to-go at a time $t$ is expressed in terms of the optimal cost-to-go at time $t + 1$. The resulting equation is called the Bellman equation that gives a formal solution to the discrete time optimal control problem. The optimal action can then directly be computed from this optimal cost-to-go.

For continuous time problems the Bellman equation becomes a partial differential equation that is called the Hamilton-Jacobi-Bellman (HJB) equation. While the bellman equation is hard to solve when the size of the state space becomes too large the HJB can in general not be solved. To approximately solve the HJB one assumes that the optimal cost-to-go is smooth, so that the HJB can be discretized which yields a bellman equation. However if the dimensionality becomes too large the number of states grows quickly (exponentially in the number of dimensions) and the problem becomes infeasible. A notable exception is when the dynamics of the system are linear and the cost is quadratic, then the HJB equation can be reduced to a system of ordinary differential equations. These ordinary differential equations are much easier to solve numerically than the HJB partial differential equations; in some cases they can even be solved exactly.

1.3.2 Kullback-Leibler control problems allow to formulate control problems as inference problems

Another special class of control problems that allows for efficient solution methods are Kullback-Leibler (KL) control problems [Tod09, BK14, KGO12]. For these kind of control problems the underlying dynamics as well as the state-dependent part of the cost may be non-linear but the control signal must be assumed to be able to fully determine the transition probabilities between states. The effect of control on the system can be limited by a control cost that is proportional to a KL-divergence that measures the deviation of the controlled transitions from some uncontrolled dynamics.

For KL-control problems one can get a closed form solution of the optimal cost-to-go, which takes the form of an expected value of the exponentiated cost under the uncontrolled dynamics. The computation of the optimal cost-to-go can then be expressed as an inference problem which allows to use solution methods originally developed for probabilistic inference problems. For discrete KL control problems, techniques originally developed for graphical-model inference problems, such as believe propagation and the cluster variation method, have been used to efficiently compute an approximate solution [KGO12]. To solve continuous KL-control problems, also called Path Integral control problems [Kap05, TK15], Monte Carlo techniques have been employed, taking advantage of the fact that the optimal cost-to-go can be expressed as an expected value and therefore be approximately estimated by sampling techniques. In the chapters 4 and 5 we give a formal introduction for the discrete and continuous case respectively.
1. Introduction

1.3.3 Solve control problems with Monte Carlo methods

The sampling approach that is used for the continuous problems has some advantages over the methods that are used for the discrete problems: It is model free, in the sense that we do not need to know the structure of the dynamical system that we want to control – we only need to be able to draw samples from it. Further the computational cost is not directly dependent on the dimensionality of the problem; it can thus in principle deal with very high dimensional problems, as long as one can get a good sample estimate of the optimal cost-to-go. However in practice the sampling efficiency is often low, which makes the naive Monte Carlo approach ineffective: Low sampling efficiency is due to the dominance of a few samples in the empirical average so that the effective sample size is low and the variance of the estimator is high. This problem can be solved with an importance sampler: a good importance sampler mainly generates samples that actually contribute to the empirical average which enhances sampling efficiency. To use this in practice a good importance sampler has to be learned or constructed for the control problem.

Adaptive importance sampling as an efficient way to solve continuous and discrete control problems

Recently an adaptive importance sampling technique that iteratively learns an importance sampler has been proposed for Path Integral control problems [KR16]. This PICE algorithm leverages the fact that the optimal importance sampler is identical to the optimal control solution for Path Integral control problems [TK15]: It utilizes a parametrized proposal controller as importance sampler that is iteratively updated to minimize the cross-entropy, an information-theoretic quantity that measures the deviation between the proposal control and a sample based estimate of the optimal controller. This results in an steadily improving sampling efficiency, allowing efficient calculation of the optimal control solution.

These efficient adaptive importance sampling techniques where originally only introduced for the continuous Path Integral control problems. In chapter 4 (based on [TGK16]) we extend this approach to discrete KL-control problems. We use these sampling based methods to control the topology of a growing network. To solve this control problem we construct a parametrized importance sampler based on features that represent the network topology. We demonstrate that this can be used to learn an efficient sampler that allows to compute an approximate control solution for this high dimensional control task.

A bootstrapping problem in adaptive importance sampling and its solution

For some problems, PICE and its discrete variant can be efficient in computing an optimal controller, but in general these methods suffer from a critical problem: In order to optimize the proposal controller the gradient of the cross-entropy has to be
computed, which is given by a weighted sample-average of the log-likelihood of the proposal control. If the proposal controller is a bad importance sampler, some of these weights can dominate and the PICE gradient cannot be estimated efficiently, yielding a problem to bootstrap the algorithm. While for some easy problems this just results in a few errant updates until the importance sampler is good enough, for harder problems the initial PICE updates are too noisy to give a good learning signal and get the adaptive importance sampling procedure started [RK17]. A heuristic solution for this problem has been suggested in [RK17]. They introduced an effective temperature that allows to artificially reduce the spread of the weights in the PICE gradient. While this reduces the variance of this gradient estimator, it gives a bias with possible unknown side effects. In [RK17] the initial bootstrapping phase, in which the application of this heuristic is necessary, takes only a few updates and afterwards the pure PICE method can be used. For more complex control problems, where the algorithm spends a majority of the optimization time in this bootstrapping phase, this heuristic solution is unsatisfactory since it is no longer clear what is actually optimized.

In chapter 5 we approached this problem. We showed that the PICE method with an effective temperature is optimizing an objective that turns out to be a smoothed version of the original cost function; and that this smoothing leaves the optimal control solution invariant. For different temperature levels, the optimization of this smoothed cost interpolates between PICE and a greedy optimization of the non-smoothed cost. Based on this we developed a new algorithm (Adaptive smoothing of Path Integral Control), that adaptively changes the level of smoothing to prevent the sampling problems that hamper the practical use of PICE. In a numerical study we show that ASPIC exhibits faster convergence than PICE or greedy cost-optimization. A theoretical analysis reveals a possible mechanism behind the advantage of ASPIC over greedy cost optimization: Local optimization of the smoothed cost anticipates a subsequent step and minimizes the cost that results from this this two-step update. It is therefore less myopic than a local optimization of the non-smoothed cost, which might explain the accelerative effect of smoothing on policy optimization.

1.4 Conclusion: Principles of prediction and control

In this thesis, we have discovered new principles of how the behaviour of a system can be predicted and developed methods that use these predictions to compute a control strategy to steer the system to a desired state.

We identified bulk-degradation processes as an important building block to predict how matter self-organizes into pattern (chapter 2), and showed how a combination of a precise spike coding scheme and the reservoir-computing learning-paradigm enables biological systems to learn from data how to predict the behaviour of another system (chapter 3). We showed how to use these predictions to compute the control of discrete systems (chapter 3.6) and continuous systems, outperforming
1. Introduction

previous methods in terms of robustness and efficiency (chapter 4).

Getting complex systems to behave in a desired manner requires powerful models that can predict them and efficient methods to turn these predictions into control strategies. This work provides building blocks and methods of prediction and control to contribute to this project.
Chapter 2

Geometry induced protein pattern formation

2.1 Abstract

Protein patterns are known to adapt to cell shape and serve as spatial templates that choreograph downstream processes like cell polarity or cell division. But how can pattern-forming proteins sense and respond to the geometry of a cell, and what mechanistic principles underlie pattern formation? Current models invoke mechanisms based on dynamic instabilities arising from nonlinear interactions between proteins but neglect the influence of the spatial geometry itself. Here we show that patterns can emerge as a direct result of adaptation to cell geometry, in the absence of dynamical instability. We present a generic reaction module that allows protein densities robustly to adapt to the symmetry of the spatial geometry. The key component is an NTPase protein that cycles between nucleotide-dependent membrane-bound and cytosolic states. For elongated cells we find that the protein dynamics generically leads to a bipolar pattern, which vanishes as the geometry becomes spherically symmetrical. We show that such a reaction module facilitates universal adaptation to cell geometry by sensing the local ratio of membrane area to cytosolic volume. This sensing mechanism is controlled by the membrane affinities of the different states. We apply the theory to explain AtMinD bipolar patterns in ΔEcMinDE E. coli. Due to its generic nature, the mechanism could also serve as a hitherto unrecognized spatial template in many other bacterial systems. Moreover, the robustness of the mechanism enables self-organized optimization of protein patterns by evolutionary processes. Finally, the proposed module can be utilized to establish geometry-sensitive protein gradients in synthetic biological systems.¹

¹This chapter is based on Dominik Thalmeier, Jacob Halatek, Erwin Frey. Geometry-induced protein pattern formation Proceedings of the National Academy of Sciences 113 (3), 548-553, 2016.
2.2 Introduction

Protein patterns serve to initiate and guide important cellular processes. A classic example is the early patterning of the *Drosophila* embryo along its anterior-posterior axis [EJ04]. Here maternal morphogen gradients initiate a complex patterning process which subsequently directs cell differentiation. However, protein patterns play a regulatory role even at the single cell level. For example, they determine cell polarity and the position of the division plane. In the yeast *Saccharomyces cerevisiae*, the GTPase Cdc42 regulates cell polarization which in turn determines the position of a new growth zone or bud site. This pattern-forming process is driven by the interaction between a set of different proteins that cycle between the plasma membrane and the cytoplasm [WSAWL03, KFWSF13]. In the rod-shaped bacterium *Escherichia coli*, Min proteins accumulate at the ends of the cell to inhibit the binding of the division proteins [Lut07, RdB99]. Here, the main player in the pattern-forming process is the ATPase MinD. It attaches to the membrane in its ATP-bound state and recruits MinE and further MinD-ATP from the cytosol [HGL02]. Cycling of proteins between membrane and cytosol is mediated by the action of MinE, which stimulates the intrinsic ATPase activity of MinD and thereby initiates its detachment. The ensuing oscillatory pattern directs the division machinery to mid-cell, enabling proper cell division in two viable daughter cells.

In all of these processes, regulatory proteins establish chemical gradients or patterns that reflect aspects of cell shape. But how is it achieved in the absence of an external template? Many possible mechanisms have been proposed and they are by no means fully classified yet [SH12, ZCT+15]. Establishing a pattern involves definition of preferred accumulation points and requires that the symmetry of the homogeneous state is broken. In *Bacillus subtilis*, there is good evidence suggesting that DivIVA recognizes negative membrane curvature directly by a mechanism which is intrinsic to this cell division protein [LHV+09, SH12]. In contrast, enrichment of MinD at the cell poles in *E. coli* is an emergent property of the collective dynamics of several proteins. As shown in Refs. [MdB01, HRdV01, MK05, HMW03, VHY08, HF12, FE06], the non-linear dynamics of the Min system leads to a polar pattern, which oscillates along the long axis and is clearly constrained by cell geometry. A clear disadvantage of such self-organized symmetry-breaking through a dynamical instability is that the kinetic parameters must be fine-tuned in order to allow the establishment of a stable polar pattern.

Here we show that cell geometry itself can enforce a broken symmetry under generic conditions without any need for fine-tuning. We introduce a class of geometry-sensing protein systems whose only stable state is a spatial pattern that is maintained by energy consumption through an ATPase or GTPase (NTPase). The proposed mechanism is based on a generic property of diffusion: The probability that a protein diffusing through the cytosol will strike (and attach) to the membrane scales with the area of membrane accessible to it. Thus, close to the poles of a
rod-shaped cell most of the trajectories available lead to the membrane. Close to mid-cell, where the membrane is almost flat, about half of the the possible paths lead away from the membrane. However, on its own, this mechanism only produces transient patterns on the membrane, as the system approaches a stable, uniform equilibrium in finite time [RLA+13]. Moreover, patterns only emerge from specific initial conditions. In this paper we ask: How can this generic property of diffusion be complemented by a minimal set of biomolecular processes to robustly maintain patterns? We show that the NTPase activity of a single protein that cycles between membrane and cytosol is sufficient to achieve this goal. Our analysis shows that an inhomogeneous density profile is established on the membrane in the generic case where the affinities of NTP and NDP-bound forms differ. Moreover, these membrane-bound patterns are amplified if the proteins are able to bind cooperatively to the membrane (e.g. due to dimerization). This mechanism is highly robust because the stable, uniform equilibrium is simply replaced by a unique, stable patterned state. In particular, the mechanism involves no dynamical instability and requires no fine-tuning of parameters.

Experimental support for the proposed mechanism comes from *E. coli* mutants in which both EcMinD and EcMinE were replaced by chloroplastic AtMinD (MinD homologue from Arabidopsis) [ZHJ+09]. With this single ATPase [AM05] the system establishes a bipolar pattern along the long axis, rescuing the \( \Delta \text{MinDE} \) mutant from cell division pathologies. Mutation studies of the Walker-A binding module show that AtMinD (unlike EcMinD) can form dimers on the membrane even in its ADP-bound form [ZSC+05, AM05, FNI+04], suggesting that both forms can cooperatively bind to the membrane. Our study shows that such cooperativity leads to a bipolar pattern along the long axis of the cell, as observed. Furthermore, we suggest that due to its generic nature, the binding module might also play an essential role in other bacterial pattern-forming systems.

### 2.3 A generic reaction module for sensing of cell geometry

We consider a reaction module comprised of a single type of NTPase which cycles between an NDP-bound (P\(_{\text{NDP}}\)) and a NTP-bound (P\(_{\text{NTP}}\)) state (Fig. 2.1A). Both forms are allowed to freely diffuse in the cytosol and the membrane with diffusion constants \( D_c \) and \( D_m \), respectively. For the biochemical reaction kinetics we assume that (i) cytosolic P\(_{\text{NDP}}\) undergoes nucleotide exchange with a rate \( \lambda \); (ii) both protein species can bind to the membrane with respective attachment rates \( \omega^{+}_P \) and \( \omega^{+}_T \); (iii) in addition to direct membrane attachment, each protein species can also bind cooperatively to the membrane, forming homodimers, with corresponding recruitment rates \( k_{dD} \) for P\(_{\text{NDP}}\) and \( k_{dT} \) for P\(_{\text{NTP}}\); (iv) hydrolysis of P\(_{\text{NTP}}\) triggers detachment with rate \( \omega^{-}_T \) which is thus converted into cytosolic P\(_{\text{NDP}}\); (v) membrane-
2. Geometry induced protein pattern formation

Bound $P_{\text{NDP}}$ is released to the cytosol with detachment rate $\omega_d^-$. For a mathematical formulation in terms of reaction-diffusion equations please refer to Eqs. (2.1)-(2.6) in the Supplement 2.7.

**Figure 2.1: Minimal reaction module for geometry-induced cell polarity.** (A) Illustration of the reaction module: Cytosolic $P_{\text{NDP}}$ can exchange its nucleotide, $P_{\text{NDP}} \rightarrow P_{\text{NTP}}$, with rate $\lambda$. $P_{\text{NTP}}$ attaches to the membrane with rate $\omega_t^+$ where it recruits further $P_{\text{NTP}}$ with rate $k_{tT}$. At the membrane, hydrolysis triggers detachment with rate $\omega_t^-$ such that membrane-bound $P_{\text{NTP}}$ is converted to cytosolic $P_{\text{NDP}}$. Cytosolic $P_{\text{NDP}}$ attaches to the membrane with rate $\omega_d^+$ where it recruits further $P_{\text{NDP}}$ with rate $k_dD$ or detaches with rate $\omega_d^-$. (B) Membrane-bound proteins accumulate either at mid-cell (left) or form a bipolar pattern with high protein densities at the cell poles (right). The left and right plot show the normalized concentration of the membrane density (blue curve) and the corresponding geometry of the cell (grey ellipse). The membrane density of the protein is divided by its minimum concentration (left: $113\mu m^{-1}$, right: $100\mu m^{-1}$) such that the minimum of the normalized density is 1. The polarity $P$ (color bar in plot is logarithmically spaced) of the pattern strongly depends on cell geometry and preference $R$ for the recruitment of a certain nucleotide state (middle); the length of the short axis is fixed at $l=1\mu m$, and we have used $k_dD+k_{tT}=0.1\mu m/s$. While for large $R$ (preferential recruitment of $P_{\text{NDP}}$) the proteins form a bipolar pattern on the membrane, the membrane-bound proteins accumulate at mid-cell for small $R$ (preferential recruitment of $P_{\text{NTP}}$). If the recruitment processes are balanced ($R=0$) the pattern is flat and polarity vanishes. The cell geometry determines how pronounced a pattern becomes: The more elongated the ellipse, the more sharply defined the pattern, while it vanishes completely when the ellipse becomes a circle.

This reaction module serves as a model for the bipolar pattern of AtMinD in *E. coli* cells [ZHJ*09]: AtMinD is an ATPase [AM05] which has been reported to dimerize [AM05, FNI*04]. This process thus provides for cooperative membrane binding. Unlike EcMinD [ZSC*05], AtMinD dimerizes even when its Walker-A binding module is inactivated [AM05], locking the protein in its ADP-bound state. This strongly suggests that also the ADP-bound form of AtMinD exhibits cooperative membrane binding, as we have assumed in the above reaction scheme by introducing a recruitment rate $k_{dD}$ for $P_{\text{NDP}}$. Overall, there is strong evidence that AtMinD shows the same interactions with the membrane as EcMinD but with additional cooperative membrane binding in its ADP-bound state.
2.4. Results

If not mentioned otherwise, we use the following model parameters, which are set to experimental values acquired for \( E. coli \) if available. The diffusion constants in the cytosol and on the membrane are set to \( D_c = 16 \mu m^2/s \) and \( D_m = 0.013 \mu m^2/s \), respectively [MRFF+06, LFFH+11]. The nucleotide exchange rate is set to \( \lambda = 6s^{-1} \) [HF12] to meet the lower bound of \( 3s^{-1} \) [MRFF+06]. The kinetic parameters, are chosen to be of the order of \( 1 \mu m/s \) for attachment, \( 1s^{-1} \) for detachment and \( 0.1 \mu m^2/s \) for recruitment [HF12]; for the specific values see Table 2.1 in the Supplement 2.7. In the numerical studies, the cell shape is modeled as a two-dimensional ellipse. (Remark: For the sake of clarity we will still use the terms of cytosolic volume and membrane area instead of areas and lines.) This reduced geometry has the same basic symmetries as the real geometry of an \( E. coli \) cell. Importantly, in contrast to a one-dimensional model, it fully accounts for the different dimensionalities of cytosol and membrane. This will turn out to be essential for the ability of the system to generate protein patterns that reflect cell geometry. The overall protein density is set to a physiologically typical value of the order of \( 1 \mu M \) [SFK+02]. For a cell which is \( 5 \mu m \) long and has a width of \( 1 \mu m \) this gives a fixed protein number of about 2000 MinD molecules. Specifically, in our numerical studies we set the protein density in the bulk to be \( \rho = 500 \mu m^{-2} \) if all proteins are in the cytosol. To accommodate changes in cell size, we keep this mean density constant and change the number of proteins as appropriate.

2.4 Results

2.4.1 The impact of cell geometry on protein gradients in elongated cells

We performed a numerical analysis of this reaction module, paying particular attention to the effect of varying the cell geometry and the degree of cooperativity in membrane binding (Fig. 2.1B). Our simulations show that in elongated cells the protein density on the membrane is always inhomogeneous and reflects the local cell geometry. Indeed, one can show analytically that the homogeneous steady state ceases to exist as one passes from circular to elliptical geometry (c.f. Supplement 2.7). We observe two distinct types of pattern: membrane-bound proteins either accumulate at mid-cell or form a bipolar pattern with high densities at both cell poles. The polarity of these patterns is quantified by the ratio of the density of membrane-bound proteins located at the cell poles (\( u_{\text{pole}} \)) to that at mid-cell (\( u_{\text{mid-cell}} \)): \( P = u_{\text{pole}} / u_{\text{mid-cell}} \). First, we investigated the impact of preferential recruitment of either \( P_{\text{NTP}} \) or \( P_{\text{NDP}} \) to the membrane, defined as \( R = (k_{dD} - k_{dT}) / (k_{dD} + k_{dT}) \), on cell polarity. We find that proteins accumulate at the cell poles (\( P > 1 \)) if there is a preference for cooperative binding of \( P_{\text{NDP}} \) (\( R > 0 \)). Moreover, the polarity \( P \) of this bipolar pattern becomes more pronounced with increasing \( R \). This scenario corresponds to the strongly bipolar pattern of AtMinD observed in mutant \( E. coli \).
2. Geometry induced protein pattern formation

cells lacking EcMinD and EcMinE [ZHJ+09]. In contrast, when cooperative binding favors \( P_{\text{NTP}} \) (\( R<0 \)), proteins accumulate at mid-cell (\( P<1 \)). Thus, the sign of the recruitment preference \( R \) for a protein in a particular nucleotide state controls the type, while its magnitude determines the amplitude of the pattern. Next, we investigated how cell geometry affects the pattern, while keeping \( R \) fixed. Upon varying the length of the long axis, \( L \), while keeping the length of the short axis fixed at \( \ell=1\mu\text{m} \), we find that the aspect ratio \( L/\ell \) controls the amplitude of the pattern, but leaves the type of pattern unchanged. With increasing eccentricity of the ellipse, the respective pattern becomes more sharply defined; for a spherical geometry the pattern vanishes. In summary, cell geometry controls the definition of the pattern, and the preference for membrane recruitment of a certain nucleotide state determines the location on the cell membrane where the proteins accumulate and how pronounced this accumulation becomes.

2.4.2 Why geometry influences patterning

Our finding that recruitment is a major determinant of cell polarity suggests that there is some underlying intrinsic affinity of the two protein species for either the cell poles or the mid-zone. This affinity can not be encoded in the attachment or recruitment rates alone, since these are position-independent. Instead, it must emerge from the interplay between these reactions, cell geometry, and diffusion. To uncover the underlying mechanism we first performed a numerical study from which all cooperative membrane binding processes were omitted, such that the dynamics became linear. Interestingly, we observed that although the overall protein density is homogeneous in the cytosol (see Supplement 2.7), \( P_{\text{NDP}} \) and \( P_{\text{NTP}} \) are nevertheless spatially segregated, accumulating in the vicinity of the cell poles and close to mid-cell, respectively (Fig. 2.2A). This observation, the origin of which will be discussed later, explains how patterns of membrane-bound proteins arise: These inhomogeneities in protein densities in the cytosol serve as seeds for the polarization of the protein pattern on the membrane and their respective impact is regulated by the attachment rates \( \omega_D^+ \) and \( \omega_T^+ \). The pattern of the protein species with the higher membrane affinity determines the type of the pattern (Fig. 2.2C). If \( P_{\text{NDP}} \) has the larger membrane affinity, a bipolar pattern emerges, whereas one observes enrichment of membrane-bound proteins at mid-cell if attachment of \( P_{\text{NTP}} \) dominates. Note that the detachment rates have the inverse effect (c.f. Supplement 2.7).

Next, to analyze the additional nonlinear effects of membrane recruitment we considered a situation, illustrated in Fig. 2.2D, where both nucleotide states have the same membrane affinity. As a result the steady-state membrane density becomes uniform (see Supplement 2.7). Since cooperative membrane binding effectively increases the affinity of a protein species just like an increase in the respective attachment rate, we expected that membrane patterns could be restored by switching
the recruitment processes back on. Indeed, we found a strong increase in polarity upon raising the recruitment rate $k_{4D}$ for fixed $k_{1T}=0$ (Fig. 2.2E). Moreover, for large recruitment rates, not only does the relative level of the two species on the membrane change, but the pattern of $P_{NDP}$ becomes highly polar (Fig. 2.2F). The reason is the positive feedback facilitated by cooperative membrane binding: In membrane regions facing a cytosolic region with an enhanced $P_{NDP}$ concentration, binding leads to a locally increased concentration which in turn increases the net attachment rate. Recruitment strongly amplifies the slight dominance of $P_{NDP}$ already existing at the cell poles in the absence of cooperative membrane binding, and thereby leads to the observed strongly bipolar $P_{NDP}$ membrane pattern.

In summary, the above analysis shows that the mechanism underlying the pattern-forming process is intrinsic to the protein dynamics: An inhomogeneous protein density in the cytosol together with unequal membrane affinities of the two forms leads to a spatially nonuniform accumulation of membrane-bound proteins. Nonlinear dynamics in the form of cooperative membrane binding (recruitment) serves to amplify these weakly nonuniform profiles into pronounced membrane patterns.

### 2.4.3 Cytosolic reaction volume determines the pattern

After investigating the phenomenology of geometry dependent pattern formation we were left with the key question: What is the origin of the observed spatial segregation of $P_{NTP}$ and $P_{NDP}$ in the cytosol? Since these patterns form without cooperative membrane binding, the mechanism must be based on the combined effect of membrane attachment and detachment, diffusion, and nucleotide exchange. Moreover, as all chemical processes are spatially uniform, the key to understanding the impact of cell geometry must lie in the diffusive coupling of these biochemical processes.

Consider the situation where the attachment rates for $P_{NDP}$ and $P_{NTP}$ are equal, such that the total protein density on the membrane becomes spatially homogeneous (see Fig. 2.2D). Only $P_{NDP}$ is released from the membrane. Hence, the latter acts as a source of cytosolic $P_{NDP}$. Because, in addition, cytosolic $P_{NDP}$ is transformed into cytosolic $P_{NTP}$ by nucleotide exchange, we have all the elements of a source-degradation process. The ensuing density profile for $P_{NDP}$ in the cytosol is exponential with the decay length set by $l_\lambda = \sqrt{D_c/\lambda}$. Due to membrane curvature these reaction volumes overlap close to the cell poles (Fig. 2.2B bottom), which implies an accumulation of $P_{NDP}$ at the cell poles. The effect becomes stronger with increasing membrane curvature. Moreover, there is an optimal value for the penetration depth $l_\lambda$, roughly equal to a third of the length $l$ of the short cell axis, that maximizes accumulation of $P_{NDP}$ at the cell poles (Fig. 2.2B top). As $l_\lambda$ becomes larger than $l$, the effect weakens, because the reaction volumes from opposite membrane sites also overlap at mid-cell. In the limit where $l_\lambda$ is much
smaller than the membrane curvature at the poles, the overlap vanishes and with it the accumulation of $P_{NTP}$ at the poles.

Expressed differently, these heuristic arguments imply that the local ratio of the reaction volume for nucleotide exchange to the available membrane surface is the factor that explains the dependence of the protein distribution on cell geometry. To put this hypothesis to the test we performed numerical simulations that are in the spirit of a minimal system approach taken by in-vitro experiments [LFFH+11, LM14]. In our numerical setup we considered a cytosolic volume adjacent to a flat membrane, as illustrated in Fig. 2.3. We were interested in how alterations in the volume of cytosol available for protein diffusion and/or nucleotide exchange would affect the density profile on the membrane.

In accordance with our hypothesis, we find that excluding volume for diffusion in the vicinity of a flat membrane reduces the available reaction volume locally and leads to accumulation of proteins at the membrane (Figs. 2.3A,C, and D). The larger the volume excluded, the more proteins accumulated at the membrane. To focus on reaction volume explicitly we considered a situation where nucleotide exchange was disabled in a given region of the cytosolic area but proteins could still diffuse in and out of it. Again, we found protein accumulation at the nearby membrane but with reduced amplitude (Fig. 2.3B). Hence, these numerical studies strongly support our heuristic arguments and lead us to conclude that it is indeed exclusion of the reaction volume for nucleotide exchange that provides for the adaptation of the pattern to the geometry of the setup. Likewise, the membrane patterning in a cell could be effected by the nucleoid if the DNA material acts as a diffusion barrier, though at present this is debated [SPL+14]. In the Supplement 2.7 we study how different sizes of effective excluded volume changes the membrane pattern. While bipolarity is still obtained for a broad parameter range, the complex geometry gives rise to a richer spectrum of possible patterns: For large sizes of excluded volume, accumulation at the poles occurs for $R<0$ (preferential recruitment for $P_{NTP}$) while for $R>0$ the proteins accumulate at mid-cell. For intermediate sizes, there are parameter ranges where patterns with several maxima, not necessarily at the poles or mid-cell, are observed.

### 2.4.4 Pattern formation does not require a dynamical instability

The above analysis shows that the difference in local reaction volume for cytosolic nucleotide exchange is the key element of the mechanism underlying geometry sensing. To put this result in perspective with pattern formation mechanisms based on dynamical instabilities we consolidated the key properties of the spatially extended model in a spatially discretized version amenable to rigorous analytical treatment (Fig. 2.4A): Diffusion in the cytosol and on the membrane is treated in terms of exchange processes between a network of nodes. A minimal set comprises
four nodes on the membrane, two at the poles and two at mid-cell, and a distribution of nodes in the cytosol which ensures that the ratio of membrane area to bulk volume at the cell poles is higher than at mid-cell. Since all observed stationary patterns are symmetrical with respect to both symmetry axes, we can further reduce the network to one quadrant of the ellipse (see Fig. 2.4B). We are now left with a network of one membrane node at a pole and one at mid-cell, two nodes serving as the interface between membrane and cytosol, and three cytosolic nodes whose distribution reflects the asymmetry in the cytosolic reaction volume between the cell poles and mid-cell.

We have analyzed the ensuing mathematical model, a system of coupled ordinary differential equations, in the context of dynamic systems theory; for mathematical details and the model parameters used please refer to the Supplement 2.7. Confirming our previous reasoning, we found that the reduced network model indeed leads to polarization between cell pole and mid-cell (Fig. 2.4B). Moreover, from a bifurcation analysis we learn that generically the dynamic system does not exhibit a bifurcation: There is only one physically possible solution with positive protein density on the membrane and this density increases with recruitment rate (Fig. 2.4C). Only in the special (non-generic) case where the attachment rate of $P_{NDP}$ vanishes, $\dot{\omega}_D^+ = 0$, do we find a transcritical bifurcation. Then, there is a critical recruitment rate $\hat{k}_{dD}^*$ below which the membrane is depleted of $P_{NDP}$. In other words, generically the system shows an imperfect transcritical bifurcation, which implies robustness of the mechanism linking protein distribution to cell geometry.
2. Geometry induced protein pattern formation

![Illustration of the source-degradation mechanism for the spatial segregation](image)

**Figure 2.2: Membrane affinity controls, and recruitment amplifies geometry adaption.** The cells used for the numerical studies have a length of $L=5\mu m$ and a width of $l=1\mu m$. (A) Even when recruitment is turned off, $P_{\text{NTP}}$ and $P_{\text{NDP}}$ form inhomogeneous density profiles in the cytosol. $P_{\text{NDP}}$ accumulates close to the poles and is depleted at mid-cell. In contrast, $P_{\text{NTP}}$ exhibits high concentration at mid-cell and a low concentration at the poles. The attachment and detachment rates are set to $1\mu m/s$ and $1s^{-1}$, respectively, which gives a penetration depth $l_{\lambda}\approx 1.6\mu m$. (B) Illustration of the source-degradation mechanism for the spatial segregation of cytosolic $P_{\text{NDP}}$ and $P_{\text{NTP}}$. All proteins that detach from the membrane are in an NDP-bound state and can undergo nucleotide exchange, the range of $P_{\text{NDP}}$ in the cytosol is limited to a penetration depth $l_{\lambda}$ (dashed lines); here $l_{\lambda}=0.35\mu m$. At the poles this reaction volumes receives input from opposing faces of the membrane resulting in an accumulation of cytosolic $P_{\text{NDP}}$ (dark red). The magnitude of this accumulation depends on the penetration depth. The polarity $P_{\text{NDP}}=u_{d}^{\text{pole}}/u_{d}^{\text{mid-cell}}$ of membrane-bound $P_{\text{NDP}}$ plotted as a function of $l_{\lambda}$ shows a maximum at $l_{\lambda}\approx 0.35\mu m$ and vanishes in the limits of large as well as small penetration depths. (C) Polarity $P$ of membrane-bound proteins as a function of the attachment rates, $\omega_{D}^{+}$ and $\omega_{T}^{+}$, with cooperative binding (recruitment) turned off. While for $\omega_{D}^{+} > \omega_{T}^{+}$ membrane-bound proteins form a bipolar pattern ($P>1$), they accumulate at mid-cell ($P<1$) for $\omega_{D}^{+} < \omega_{T}^{+}$. (D) Density profiles of membrane-bound proteins in the limit where the attachment rates of the two species are equal, $\omega_{D}^{+} = \omega_{T}^{+} = 1\mu m/s$, and recruitment is switched off. The membrane profile of the total protein density (green) is flat, while membrane-bound $P_{\text{NTP}}$ (blue) accumulates at mid-cell and $P_{\text{NDP}}$ (red) forms a bipolar pattern. (E) Polarity $P$ of the membrane-bound proteins as a function of $k_{\text{dD}}$ for $\omega_{D}^{+} = \omega_{T}^{+}$. Increasing the recruitment rate restores polarity. (F) Density profiles of membrane-bound proteins for $k_{\text{dD}}=0.1\mu m^2/s$. The density of $P_{\text{NDP}}$ (red) as well as the overall protein density (green) exhibit strongly bipolar patterns, which are much more pronounced than the corresponding patterns in the absence of cooperative membrane binding. The density of $P_{\text{NTP}}$ (blue) is comparatively flat, and there are much less membrane-bound proteins in this nucleotide state than in the $P_{\text{NDP}}$ state. The overall protein pattern is strongly dominated by $P_{\text{NDP}}$. 

18
Figure 2.3: Two-dimensional planar geometry with cytosolic volume (blue) above a membrane at $y=0$. The left, right and top boundaries of the cytosolic regime are reflecting boundaries. Diffusion and nucleotide exchange rates are set to their standard values, the total number of proteins is set to $N=50$. Black boxes indicate areas which are not accessible to the proteins and thereby generate excluded reaction volumes; the boundaries of the boxes are assumed to be reflecting. Solid curves show the normalized density of $P_{NDP}$ bound to the membrane: $\tilde{u}_d = u_d/u_{d}^{\text{max}}$. Generally, $P_{NDP}$ accumulates at membrane regions in the vicinity of the cytosolic areas with excluded reaction volumes with the effect being stronger with larger excluded reaction volumes and closer to the membrane (A,C,D). In (B) the white box indicates that within its volume all proteins are allowed to diffuse but they do not undergo nucleotide exchange. This has a similar but weaker effect to that observed in the other panels: The proteins accumulate at the membrane near the excluded reaction volume. The parameters used in these numerical experiments are summarized in Table 2.1 in the Supplement 2.7.
2. Geometry induced protein pattern formation

Figure 2.4: Reduced network model and bifurcation analysis. (A) The full spatio-temporal dynamics in an ellipse is reduced to the nonlinear dynamics of a network of coupled nodes. We take the minimal possible number of nodes reflecting the asymmetry in the ratio of membrane area to bulk volume at the cell poles and mid-cell. Diffusion in the cytosol is modeled as particle exchange processes between the nodes. The network equations are derived from a discrete time jump process. Since the symmetry of the pattern reflects the symmetry of the ellipse there is no flux of particles through either midplanes (red dashed lines). Therefore, the network can be further reduced to a single quadrant (black) with the other quadrants (gray) simply mirroring its behaviour. (B) The reduced model comprises two membrane nodes at the pole and at mid-cell, two border nodes connecting membrane and cytosol, as well as three cytosolic nodes. Node 1a captures the increased ratio of membrane area to bulk volume at mid-cell. As illustrated by the colored nodes, this minimal network model polarizes: the density of P\(_{\text{NDP}}\) at the pole node is higher than on the membrane node. (C) Density of P\(_{\text{NDP}}\) at the pole node as a function of the recruitment rate \(k_{dD}\). The density is given relative to the total number of particles located at the membrane node of the pole. For vanishing attachment rate of P\(_{\text{NDP}}\), \(\omega_D^+=0\), there is a transcritical bifurcation at a critical rate \(k_{dD}^{*} \approx 8.7 \text{s}^{-1}\) where a polarized state exchanges stability with an unpolarized state (black lines). In contrast, for the generic case of finite membrane attachment, \(\omega_D^+ > 0\), there is only one positive fixed point solution which is always stable (blue lines).
2.5 Discussion

How does protein patterning adapt to cell geometry? Dynamic models for pattern formation often reduce the cytosolic volume to the same dimension as the membrane and focus on the role of non-linear protein interactions; see e.g. Ref. [AB10, HRdV01]. At first sight this appears to make sense, since diffusion coefficients are generically much higher in the cytosol than on the membrane. Indeed, if only attachment and detachment processes are involved, any transient geometry-dependent pattern is rapidly washed out [RLA+13].

Here we have shown that the assumption of a well-mixed cytosolic protein reservoir becomes invalid as soon as cytosolic processes like nucleotide exchange, which alter protein states become involved. We have introduced a minimal reaction module with a single NTPase that cycles between membrane and cytosol. The fact that cytosolic nucleotide exchange may take place on a diffusive length scale far below cell size has been noted previously [HMW03], and it has been show that this can be critical for robust, intracellular pattern formation [HF12]. Our analysis reveals that nucleotide exchange leads to an inhomogeneous protein distribution in the cytosol, which is stably maintained and depends strongly on the geometry of the cytosolic space. As a consequence, proteins accumulate on certain membrane regions, depending on the local ratio of membrane area to cytosolic volume. In an elongated cell this serves as a robust mechanism for proper cell division by facilitating protein accumulation at the poles. The proposed reaction module operates through implicit curvature sensing and does not require that the relevant protein themselves respond to membrane curvature [SH12, LHV+09] or lipids [HMW06]. The degree and the axis of polarization depend on the level of cooperativity in membrane binding, which can be regulated by enzymes.

Our theoretical analysis suggests that evolutionary tuning of this simple reaction module is feasible: Because there is no threshold involved, polarity can be improved stepwise starting from any parameter configuration. This lack of a threshold can at the same time also be a disadvantage: without a trigger pattern formation is difficult to induce as response to an upstream event. Another distinctive element of the mechanism is the lack of a characteristic length scale (e.g. as striped Min-patterns in E. coli), instead the pattern scales with the size of the cell. Depending on the functional role this might be desired or disadvantageous.

The reaction module gives a possible explanation for the bipolar patterns of AtMinD observed in mutant E. coli cells [ZHJ+09]. Several experimental tests could be performed to validate the proposed reaction module: One route would be to study spherical E. coli cells. For this geometry, we predict that the polarization of AtMinD should vanish, since the membrane curvature is uniform. This, however, would also be the case if the kinetics of AtMinD binding is directly dependent on membrane curvature, as in the case of DivIVa [SH12, LHV+09]. To rule out this scenario an in-vitro experiment could be conducted, as described in Fig. 2.3.
2. Geometry induced protein pattern formation

In-vitro experiments might also serve as a proof of concept for the use of the suggested reaction module in nano-scale self-organization. By enzymatically regulating the kinetic rates of the process one could induce protein patterns on a membrane which then serve as templates for the localization of nano-scale structures, e.g. similar to the formation of actin cables close to Cdc42 protein caps in yeast. Localization could either be self-organized or target specific curvatures or be externally controlled by volume exclusion in the cytosolic space. If, in addition, such nano-structures exert forces on the membrane this self-organization principle could be used to regulate the shape of membranes. Thus the proposed minimal module might serve as a core network for the design of other geometry sensing protein networks.

On a more speculative note, geometry sensing protein networks like the one discussed here would enable a cell to gradually optimize its biological function, since the underlying mechanism does not involve a bifurcation threshold. For example, one could envision a biochemical network containing a protein which is able to trigger hydrolysis-driven detachment. Such a catalytic process could act selectively on the $P_{\text{NTP}}$ or the $P_{\text{NDP}}$ species. This would create an imbalance between the effective membrane affinities of $P_{\text{NTP}}$ and $P_{\text{NDP}}$, and thus regulate polarity. Moreover, the copy number of such a catalyst would become an evolutionary tunable modulator of the effective imbalance. MinE in E. Coli which stimulates the hydrolysis of membrane bound MinD-ATP is a possible instance of such a factor.

Finally, due to its generic nature the proposed mechanism might be involved in many bacterial pattern-forming systems. For instance, the sensitivity to cytosolic reaction volume provides a way to sense large cytosolic structures. This could, for instance, be part of the mechanism that guides PomZ to mid-cell in *M. xanthus* [TLSA14]. One could also imagine direct feedback mechanisms between force-exerting proteins that regulate cell shape (e.g. FtsZ ring contraction) and proteins that adapt to local cell shape by sensing the local reaction volume, and which guide the downstream accumulation of further force-exerting proteins. In this scenario cell shape could be controlled (even in a self-organized fashion) by balancing these two processes.

### 2.6 Computational methods and initial conditions

The model is mathematically described as a set of reaction diffusion equations (see Supplement 2.7). All simulations were performed with finite-element methods on a triangular mesh using Comsol Multiphysics 4.3. As initial condition all proteins where in the NDP state and located in the bulk of the ellipse. In Fig. 2.3, the particles are initially located on the membrane in the NDP state. For Figs. 2.1 and 2.2 the simulation time was 1000s, and for Fig. 2.3 it was 2000s. A steady state is reached after approximately 100s.
2.7 Supplement: Geometry induced protein pattern formation

2.7.1 Model equations

Using orthogonal elliptical coordinates (for mathematical details please refer to the following section), given by the normal \( \mu \) and tangential \( \nu \) components at the boundary as in Ref. [HF12], the reaction module is mathematically described by the following set of reaction-diffusion equations:

\[
\begin{align*}
\partial_t u_T &= D_c \Delta u_T + \lambda u_D, \quad (2.1) \\
\partial_t u_D &= D_c \Delta u_D - \lambda u_D, \quad (2.2) \\
\partial_t u_t &= k_{tT} u_t u_T + \left( \omega_t^+ u_T - \omega_t^- u_t \right) + D_m \Delta u_t, \quad (2.3) \\
\partial_t u_d &= k_{dD} u_d u_D + \left( \omega_d^+ u_D - \omega_d^- u_d \right) + D_m \Delta u_d. \quad (2.4)
\end{align*}
\]

Here \( u_D \) and \( u_T \) denote for the bulk concentrations of \( \text{P}_{\text{NDP}} \) and \( \text{P}_{\text{NTP}} \), respectively, and \( u_d \) and \( u_t \) the membrane concentrations of \( \text{P}_{\text{NDP}} \) and \( \text{P}_{\text{NTP}} \), respectively. The equations account for particle conservation. The exchange of particles between the cytosol and the membrane is determined by a reactive boundary condition [HF12, LR05] stating that the reactions equal the flux onto and off the membrane

\[
\begin{align*}
D_c \nabla_{\mu} u_T &= -k_{tT} u_t u_T - \omega_t^+ u_T, \quad (2.5) \\
D_c \nabla_{\mu} u_D &= -k_{dD} u_d u_D - \omega_d^+ u_D + \omega_d^- u_d + \omega_t^- u_t. \quad (2.6)
\end{align*}
\]

For Fig.2.3 we use orthogonal cartesian coordinates with \( (\mu, \nu) \) replaced by \( (y, x) \).

2.7.2 Orthogonal Elliptical Coordinates and Differential Operators

The model equations of the diffusion-reaction system (Eqs. (2.1)-(2.4)) with reactive boundary conditions is formulated in elliptical coordinates \( \mu \) and \( \nu \) (Fig. 2.5). In the following we introduce this coordinate system and give the differential operators used in the model equations (see also [HF12]).

For an ellipse with major semi-axis \( r_a \), minor semi-axis \( r_b \), and linear eccentricity \( d = \sqrt{r_a^2 - r_b^2} \), we choose orthogonal elliptical coordinates given by

\[
\begin{align*}
x &= d \cosh \mu \cos \nu, \quad (2.7) \\
y &= d \sinh \mu \sin \nu, \quad (2.8)
\end{align*}
\]

with \( \mu > 0 \) and \( 0 \leq \nu \leq 2\pi \).

To obtain the differential operators in a curved orthogonal coordinate system one needs the lengths of the basis vectors, \( h_\mu \) and \( h_\nu \), also called scale factors; see
2. Geometry induced protein pattern formation

Figure 2.5: Orthogonal elliptical coordinates. We use elliptical coordinates given by the normal $\mu$ and tangential $\nu$ components at the boundary. Ellipses are obtained at constant $\mu=\mu_0$.

e.g. [Spi59]. In elliptical coordinates they are given by

$$h_\mu = h_\nu = d \sqrt{\sinh^2 \mu + \sin^2 \nu}.$$  \ \ \ (2.9)

Then the gradient operator in $\mu$-direction reads

$$\nabla_\mu = \frac{1}{h_\mu} \mathbf{e}_\mu \partial_\mu = \frac{1}{d \sqrt{\sinh^2 \mu + \sin^2 \nu}} \mathbf{e}_\mu \partial_\mu.$$  \ \ \ (2.10)

with $\mathbf{e}_\mu$ being the basis vector for the $\mu$ direction. The corresponding diffusion
2.7. Supplement: Geometry induced protein pattern formation

operator (see [Spi59] page 137) in the cytosol is given by

$$\Delta = \frac{1}{h_\mu h_\nu} (\partial_\mu^2 + \partial_\nu^2)$$

$$= \frac{1}{d^2(\sinh^2 \mu + \sin^2 \nu)} (\partial_\mu^2 + \partial_\nu^2). \quad (2.11)$$

Diffusion on the cell membrane is constrained to a fixed value of $\mu$: $\mu_0 = \arctanh(r_b/r_a)$.
We formulate the diffusion operator on the membrane $\Delta_\nu$ in arclength parametrization $s(\nu)$:

$$s(\nu) = \int_0^\nu d\tilde{\nu} \sqrt{r_b^2 + (r_a^2 - r_b^2) \sin^2 \tilde{\nu}}. \quad (2.12)$$

The Laplacian then simply becomes

$$\Delta_\nu = \partial_s^2. \quad (2.13)$$

For Fig.2.3 we used cartesian coordinates such that the diffusion operators are
given by $\Delta = \partial_x^2 + \partial_y^2$ for the cytosol, and $\Delta = \partial_x^2$ for the membrane.

2.7.3 Supplementary details for the Figures

The parameters for Figs.2.1-2.3 can be found in Table 2.1.

Supplementary details for Figure 2.1

For Fig.2.1, the recruitment rate $k_{D\phi}$ was sampled in the range $0 - 0.1 \mu m^2/s$ with
50 uniformly spaced parameter values. $k_{T}$ was chosen as $k_T = 0.1 \mu m^2/s - k_{D\phi}$
such that $k_{D\phi} + k_T = 0.1 \mu m^2/s$. The major semi-axis of the ellipse was varied in the
range from 0.5 $\mu m$ to 2.5 $\mu m$ with 50 equally spaced parameter values. The minor
semi-axis was kept constant at 0.5 $\mu m$.

Supplementary details for Figure 2.2

For Fig.2.2 we used an elliptic geometry for the cell with a length of 5 $\mu m$ and a
width of 1 $\mu m$. For Fig.2.2B, the penetration depth $l_\lambda$ was varied between $10^{-3}$
and $10^7$; the values are equally spaced on a logarithmic scale. For Fig.2.2C, the
parameters $\omega_D^+$ and $\omega_T^+$ were varied between 0.11/s and 11/s with 50 uniformly
spaced parameter values.
2. Geometry induced protein pattern formation

Supplementary details were varied in the corresponding figure, details are found in the corresponding section.

Parameters used to create the simulations used in Figs. 2.1-2.3. Cells marked with a "-" denote that this parameter was varied. Details are found in the corresponding section.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
<td>0.013</td>
<td>0.013</td>
<td>0.013</td>
<td>0.013</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
<td>1.6</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$\omega$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table 2.1:** Parameters used to create the simulations used in Figs. 2.1-2.3. Cells marked with a "-" denote that this parameter was varied.
2.7. Supplement: Geometry induced protein pattern formation

Supplementary details for Figure 2.3

The quadratic boxes in Fig.2.3 have side length 0.5\(\mu\)m. In panel A and B, the position of the lower left point of the square is at \(x=1.75\mu\)m and \(y=0.05\mu\)m. In panel C the position of the left box is also \(x=1.75\mu\)m and \(y=0.05\mu\)m, while the position of the right box is \(x=4\mu\)m and \(y=0.15\mu\)m. In panel D, the position of the left box is also \(x=1.75\mu\)m and \(y=0.05\mu\)m; the right box has side lengths 0.5\(\mu\)m and 0.1\(\mu\)m, its lower left point is at \(x=4\mu\)m and \(y=0.05\mu\)m.

Supplementary details for Figure 2.4

The parameters for the bifurcation plot Fig.2.4D can be found in Table 2.2.

<table>
<thead>
<tr>
<th>Figure</th>
<th>N</th>
<th>(\lambda)</th>
<th>(\hat{\omega}_D^+)</th>
<th>(\hat{\omega}_d^-)</th>
<th>(\hat{k}_{dD})</th>
<th>(\hat{\omega}_T^-)</th>
<th>(\hat{\omega}_T^+)</th>
<th>(\epsilon)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4B</td>
<td>1</td>
<td>1 s(^{-1})</td>
<td>1 s(^{-1})</td>
<td>10 s(^{-1})</td>
<td>100 s(^{-1})</td>
<td>1 s(^{-1})</td>
<td>1 s(^{-1})</td>
<td>1 s(^{-1})</td>
</tr>
<tr>
<td>2.4C</td>
<td>1</td>
<td>1 s(^{-1})</td>
<td>0 s(^{-1}) and 0.05 s(^{-1})</td>
<td>1 s(^{-1})</td>
<td>-</td>
<td>1 s(^{-1})</td>
<td>1 s(^{-1})</td>
<td>1 s(^{-1})</td>
</tr>
</tbody>
</table>

Table 2.2: Parameters used in Fig.2.4. Cells marked with a "-" denote that this parameter was varied in the corresponding figure, details are found in the corresponding Supplementary details of section.

2.7.4 Equations of the minimal system

To frame the coarse-grained model we use the following nomenclature: The quantities \(u_A^X\) denote the number of the proteins in state \(A\), which may be cytosolic P\(_{NDP}\) \((A=D)\), cytosolic P\(_{NTP}\) \((A=T)\), membrane bound P\(_{NDP}\) \((A=d)\) or membrane bound P\(_{NTP}\) \((A=t)\). The superscript \(X\) signifies the spatial position in the coarse-grained network. Here, membrane nodes are denoted with \(m\) for mid-cell and \(p\) for pole, respectively. The neighboring cytosolic nodes of these membrane nodes are also denoted by \(m\) and \(p\). The remaining nodes in the bulk of the cytosol are denoted by 1, 1\(\alpha\) and 2 (see Fig.2.4B).

The dynamic processes for the membrane nodes in the coarse-grained model comprise, as in the spatially extended model, attachment and detachment as well as recruitment processes. The ensuing nonlinear equations read

\[
\partial_t u^m_d = \hat{k}_{dD} u^m_d u^m_d + \hat{\omega}_D^+ u^m_d - \hat{\omega}_d^- u^m_d, \quad (2.14)
\]

\[
\partial_t u^p_d = \hat{k}_{dD} u^p_d u^p_d + \hat{\omega}_D^+ u^p_d - \hat{\omega}_d^- u^p_d, \quad (2.15)
\]

\[
\partial_t u^p_t = -\hat{\omega}_T^- u^p_t + \hat{\omega}_T^+ u^p_t, \quad (2.16)
\]

\[
\partial_t u^m_t = -\hat{\omega}_T^- u^m_t + \hat{\omega}_T^+ u^m_t, \quad (2.17)
\]
2. Geometry induced protein pattern formation

where a hat on the rates indicates that the rates are for the discrete model. For the sake of simplicity we omit in the minimal model the cooperative membrane binding of the P_{NTP} species, since this process is not necessary to obtain a bipolar pattern. In section 2.7.7 we will perform a continuum limit and show how these rates are related to the corresponding rates of the continuum model. For the bulk cytosolic nodes we have

\[
\begin{align*}
\partial_t u^1_D &= -\epsilon (3u^1_D - u^m_D - u^{1a}_D - u^2_D) - \lambda u^1_D, \\
\partial_t u^1_T &= -\epsilon (3u^1_T - u^m_T - u^{1a}_T - u^2_T) + \lambda u^1_T, \\
\partial_t u^{1a}_D &= -\epsilon (u^{1a}_D - u^1_D) - \lambda u^{1a}_D, \\
\partial_t u^{1a}_T &= -\epsilon (u^{1a}_T - u^1_T) + \lambda u^{1a}_T, \\
\partial_t u^2_D &= -\epsilon (2u^2_D - u^1_D - u^m_D) - \lambda u^2_D, \\
\partial_t u^2_T &= -\epsilon (2u^2_T - u^1_T - u^m_T) + \lambda u^2_T,
\end{align*}
\]

where \( \epsilon \) is the hopping rate between nodes, and \( \lambda \) the nucleotide exchange rate in the cytosol. Finally, the dynamics for the cytosolic nodes neighboring the membrane nodes are given by

\[
\begin{align*}
\partial_t u^p_D &= -\epsilon (u^p_D - u^2_D) + \hat{\omega}_t^+ u^p_T \\
&\quad - (k_{dp} u^p_d u^p_D + \hat{\omega}_d^+ u^p_D - \hat{\omega}_d^- u^p_D), \\
\partial_t u^m_D &= -\epsilon (u^m_D - u^1_D) + \hat{\omega}_t^- u^m_T \\
&\quad - (k_{dp} u^m_d u^m_D + \hat{\omega}_d^+ u^m_D - \hat{\omega}_d^- u^m_D), \\
\partial_t u^p_T &= -\epsilon (u^p_T - u^2_T) - \hat{\omega}_t^+ u^p_T, \\
\partial_t u^m_T &= -\epsilon (u^m_T - u^1_T) - \hat{\omega}_t^+ u^m_T.
\end{align*}
\]

In addition, since the total number of particles is conserved, we have the constraint

\[
N = u^m_d + u^p_d + u^m_T + u^m_T + u^{1a}_D + u^{1a}_T + u^1_D + u^{1a}_D + u^2_D + u^2_T + u^m_T + u^m_T.
\]

To analyze the fixed point structure of this minimal system (see Fig.2.4C) we set all time derivatives in Eqs.(2.14-2.27) to zero and solved the corresponding system of equations for the particle occupation numbers \( u^x_A \). To solve the systems of equations we used Mathematica 10.0. Next we performed a linear stability analysis of those fixed points for the two cases \( \hat{\omega}_D^-=0 \) and \( \hat{\omega}_D^+>0 \).

In both cases the system has three fixed points. The main observation is that for \( \hat{\omega}_D^-=0 \) there is a transcritical bifurcation from a homogeneous P_{NDP} distribution on
the membrane to a polarized $P_{NDP}$ distribution when varying $\hat{k}_{dD}$; while for $\hat{\omega}_D^+ > 0$ the stability of the fixed points does not change and the polarized state is always stable (see Fig.2.4B).

Interestingly in the case of $\hat{\omega}_D^+ > 0$ the polarized state is the only fixed point with only positive particle occupation numbers $u_X^\mu$. Thus the other two fixed points cannot be reached when starting with an initial condition with only positive particle occupation numbers $u_X^\mu$ which leaves the polarized fixed point as the only physically possible steady state solution of the system.

### 2.7.5 Balance of membrane affinities

In this section we show how equal membrane affinities of the $P_{NDP}$ and $P_{NTP}$ species lead to the reestablishment of a spatially uniform steady state. To this end we consider the dynamics of the linear diffusion-reaction equations where the recruitment rates are set to zero, $k_{dD}=k_{tT}=0$. We will specify the conditions leading to spatially uniform protein densities in the cytosol and on the membrane.

The diffusion-reaction equations in the cytosol read

\begin{align}
\partial_t u_T &= D_c \Delta u_T + \lambda u_T, \\
\partial_t u_D &= D_c \Delta u_D - \lambda u_D,
\end{align}

(2.29, 2.30)

Because of the nucleotide exchange term, the density distribution of a protein in a given nucleotide state can not be spatially homogeneous, quite independent from the boundary conditions at the membrane. In contrast, since the equation for the total protein density, $u_c = u_T + u_D$, reads

\[ \partial_t u_c = D_c \Delta u_c , \]

(2.31)

a spatially uniform density $u_c$ in the cytosol is possible. To see what conditions need to be satisfied for that to be the case, consider the boundary conditions specifying the coupling between the membrane and the cytosol

\begin{align}
D_c \nabla_{\mu} u_T &= -\omega^+_T u_T, \\
D_c \nabla_{\mu} u_D &= -\omega^+_D u_D + \omega^-_d u_d + \omega^-_t u_t.
\end{align}

(2.32, 2.33)

Note that these boundary conditions are constant along the membrane only if the normal derivatives are. In general, this is not the case (We will discuss the case of broken circular geometry in the next section). However, reactive boundary conditions do admit stationary solutions that are uniform along the membrane if the normal fluxes vanish altogether, i.e. if the cytosolic density in steady state is constant.

Upon adding both boundary conditions (2.32) and (2.33) one obtains

\[ D_c \nabla_{\mu} u_c = -\omega^+_T u_T - \omega^+_D u_D + \omega^-_d u_d + \omega^-_t u_t . \]

(2.34)
2. Geometry induced protein pattern formation

Hence, there is a reflective boundary condition, \( \nabla \mu u_c = 0 \), and thereby a homogeneous cytosolic total protein density \( u_c \), if the following balance equation is satisfied

\[
\omega_T^+ u_T + \omega_D^+ u_D = \omega_D^- u_d + \omega_T^- u_t .
\]  

(2.35)

Inversely, this condition is satisfied if the cytosolic total protein \( u_c \) density is homogeneous. Hence, the question whether a spatially uniform cytosolic protein density is a steady state of the nonlinear dynamics reduces to the question whether the balance equation, Eq.2.35, is consistent with the stationary solutions of the diffusion-reaction equations on the membrane:

\[
\begin{align*}
\partial_t u_t &= D_m \Delta u_t + \omega_T^+ u_T - \omega_T^- u_t , \\
\partial_t u_d &= D_m \Delta u_d + \omega_D^+ u_D - \omega_D^- u_d .
\end{align*}
\]  

(2.36), (2.37)

First, let us neglect membrane diffusion. If the membrane diffusion constant would be zero, \( D_m = 0 \), the steady state conditions read

\[
\begin{align*}
0 &= \omega_T^+ u_T - \omega_T^- u_t , \\
0 &= \omega_D^+ u_D - \omega_D^- u_d ,
\end{align*}
\]  

(2.38), (2.39)

and hence the balance equation is satisfied. This implies that the overall protein density in the cytosol is spatially uniform. However, this does not imply that the membrane density is homogeneous as well. To the contrary, upon inserting \( u_c = u_D + u_T \) into Eqs.2.38 and 2.39 one obtains

\[
\begin{align*}
u_t &= \frac{\omega_T^+}{\omega_T^-} u_T, \\
u_d &= \frac{\omega_D^+}{\omega_D^-} (u_c - u_T).
\end{align*}
\]  

(2.40), (2.41)

and hence for the overall protein density on the membrane

\[
u_m = \frac{\omega_D^+}{\omega_D^-} u_c + \left( \frac{\omega_T^+}{\omega_T^-} - \frac{\omega_D^+}{\omega_D^-} \right) u_T .
\]  

(2.42)

Since \( u_T \) is always spatially heterogeneous (for \( \lambda \neq 0 \) and a non-spherical cell geometry), the protein density on the membrane can become uniform only if the membrane affinities for the two different nucleotide states balance each other\(^2\)

\[
\frac{\omega_T^+}{\omega_T^-} = \frac{\omega_D^+}{\omega_D^-} .
\]  

(2.43)

\(^2\)Note that the fractions \( \omega_T^+ / \omega_T^- \) and \( \omega_D^+ / \omega_D^- \) define the ratio of membrane to cytosolic densities for the two nucleotide states, see Eq.2.40 and Eq.2.41.
If the diffusion constant on the membrane is finite, $D_m > 0$, spatially uniform membrane densities in the cytosol and on the membrane are possible for the special case where $\omega^+_i = \omega^+_j = \omega^+$ and $\omega^-_i = \omega^-_j = \omega^-$. Then one also obtains a diffusion-reaction equations for overall protein density on the membrane

$$
\partial_t u_m = D_m \Delta u_m + \omega^+ u_c - \omega^- u_m .
$$

(2.44)

with the boundary condition for the membrane-cytosol coupling

$$
D_c \nabla \mu u_c = \omega^+ u_c - \omega^- u_m ,
$$

(2.45)

For the steady state, the overall protein density on the membrane as well as in the cytosol become uniform if

$$
\omega^+ u_c = \omega^- u_m .
$$

(2.46)

i.e. there is detailed balance between the overall membrane and cytosolic protein densities. This corresponds to the situation shown in Fig.2.2C, where we observe that $P = 1$ if the attachment rates and detachment rates of the two nucleotide states balance each other precisely ($\omega^+_i = \omega^+_j = \omega^+$ and $\omega^-_i = \omega^-_j = \omega^-$).

If however recruitment is switched on one typically does not get a spatially homogeneous solution (see Fig.2.2E for the membrane and Fig. 2.6 for the cytosol).

Remark: Since the diffusion dynamics on the membrane is in general very slow as compared to attachment and detachment processes, we expect that the diffusion-reaction dynamics on the membrane is dominated by the reaction terms. Therefore, already Eq.2.43 should be a good criterion for a spatially uniform overall protein density on the membrane.

### 2.7.6 Breaking the homogenous steady state in the circular limit

In this section we take a closer look at the cytosolic steady state in the circular limit. In coordinate free notation, the stationary cytosolic equations read

$$
0 = D_c \Delta u_T + \lambda u_D ,
$$

(2.47)

$$
0 = D_c \Delta u_D - \lambda u_D .
$$

(2.48)

As mentioned earlier $u_c = u_D + u_T$ decouples both equations as the combined species $u_c$ satisfies the Laplace equation

$$
0 = D_c \Delta u
$$

(2.49)

which always admits a constant (i.e. spatially homogenous) solution for no flux boundary conditions irrespective of the specific boundary geometry. Hence, whether the system admits a homogenous solution depends on the stationary diffusion-degradation equation (2.48) which we will now consider in elliptical coordinates.
2. Geometry induced protein pattern formation

\[ u_D = u_D(\mu, \nu) \text{ as outlined in section 2.7.2.} \]

With an ansatz employing separation of variables \( u_D(\mu, \nu) = R(\mu)\Psi(\nu) \) one obtains the well known Mathieu equations:

\[
\begin{align*}
0 &= \Psi'' + (c - 2q \cos 2\nu)\Psi \\
0 &= R'' - (c - 2q \cosh 2\mu)R,
\end{align*}
\]

where \( c \) is a constant of separation and \( q \) a dimensionless parameter given by

\[ q = -\frac{\lambda d^2}{4D_c}. \]

Note that \( q \) vanishes in the circular limit \( d \to 0 \), or if the cytosolic degradation length scale \( \sqrt{D_c/\lambda} \) becomes infinite. The solutions of the Mathieu equations are the Mathieu functions which form an orthonormal basis for the diffusion-degradation problem in elliptical geometry, c.f. the supplementary material in Ref. [HF12] and references therein. Note that only in the limit \( q \to 0 \) the angular Mathieu equation (2.50) admits a constant solution \( \Psi_0(\nu) = \Psi_0 = \text{const} \). For small \( q \), i.e. in the nearly circular case, one can express the Mathieu functions perturbatively in powers of \( q \) [HF12]. For the expansion of the homogenous solution one obtains

\[ \Psi_0(\nu) \approx 1 - \frac{q}{2} \cos(2\nu) + \mathcal{O}(q^2), \]

which has the characteristic symmetry of the bipolar patterns we observe. The amplitude of this new base state scales with \( q \) in the circular limit. Of course, since the Mathieu functions form an orthonormal basis we can construct a solution that is constant at the boundary. However, one finds that the normal derivative of this solution is not constant in angular direction (see supplementary material of [HF12] for the detailed derivation) and as such it is incompatible with the boundary conditions (2.33) that assume spatially uniform densities at the membrane. Hence, a spatially uniform solution at the membrane is lost as steady state as soon as circular geometry is broken.

2.7.7 Continuum limit of the boundary conditions of the minimal system

In this section we show how the reactive boundary conditions, as given by Eqs.(2.1)-(2.6), can be obtained as a continuum limit from the boundary conditions used in the coarse-grained network model in Eqs.(2.24-2.27). For simplicity, we will illustrate the calculation for a one-dimensional model with only one protein species (Fig.2.7), and refer the reader for a more elaborate description of the continuum limit to [EC07].

The proteins are performing a symmetric random walk on this one-dimensional grid with hopping rate \( \epsilon \). Hence, the dynamics for the occupation number \( u^x_c \) of
2.7. Supplement: Geometry induced protein pattern formation

each node \( x \) in the bulk is given by

\[
\partial_t u^x_c = \epsilon \left( u^x_c + a u^x_c - 2 u^x_c \right),
\]

(2.54)

where \( a \) is the distance between the nodes. Furthermore, we assume that the bulk is reactively coupled (by attachment and detachment processes) with the membrane node via the boundary node at position \( x=0 \):

\[
\begin{align*}
\partial_t u_m &= \hat{\omega}^+ u^0_c - \hat{\omega}^- u_m, \\
\partial_t u^0_c &= \epsilon (u^a_c - u^0_c) - \hat{\omega}^+ u^0_c + \hat{\omega}^- u_m.
\end{align*}
\]

(2.55, 2.56)

Here \( u_m \) is the occupation number of the membrane node, and \( \hat{\omega}^- \) and \( \hat{\omega}^+ \) signify the detachment and attachment rate, respectively.

In the continuum limit \( a \rightarrow 0 \), the number of grid nodes increases, and concomitantly its occupation number \( u^x_c \) decreases such that the particle (line) density \( u_c(x) = u^x_c / a \) remains finite. Then, the bulk dynamics, Eq.(2.54), becomes a diffusion equation

\[
\partial_t u_c(x) = D_c \partial_x^2 u_c(x)
\]

(2.57)

with the macroscopic diffusion constant \( D_c \) and the hopping rate \( \epsilon \) related by \( D_c = \epsilon a^2 \). Note that in the continuum limit, the occupation number of the membrane node, \( u_m \), remains a zero-dimensional quantity. It constitutes a reservoir of particles. Therefore, in the continuum limit, Eq.(2.55) simply becomes

\[
\partial_t u_m = \omega^+ u_c(0) - \omega^- u_m,
\]

(2.58)

where we have defined the macroscopic attachment rate as \( \omega^+ = \hat{\omega}^+ a \), and the macroscopic detachment rate as \( \omega^- = \hat{\omega}^- \). Finally, the equation for the boundary node, Eq.(2.56), in the continuum limit reduces to

\[
-D_c \partial_x u_c(0) = -\omega^+ u_c(0) + \omega^- u_m.
\]

(2.59)

where we have used that \( \lim_{a \rightarrow 0} \partial_t u_c(0) = 0 \); since the time scale on the boundary scales with \( a \) the node right next to the membrane is in equilibrium with the membrane. The reactive boundary condition, Eq.(2.59), states that the diffusive flux on and off the membrane \(-D_c \partial_x u_c(0)\) is balanced by attachment and detachment processes \(-\omega^+ u_c(0) + \omega^- u_m\). Note that in the boundary equations (2.5) and (2.6) the gradient \( \nabla \mu \) is the normal derivative pointing towards the membrane. Here the corresponding normal derivative is \(-\partial_x\), which also points towards the membrane.

Remark: For the elliptic geometry mapping of microscopic to macroscopic rates becomes position dependent since the volumina represented by the different grid nodes are not uniform. The minimal model in Fig.2.4 is not meant to exactly map back to the elliptic geometry in a continuum limit, rather to illustrate and analyze the effect of an inhomogeneous geometry on the reaction diffusion dynamics in a minimal model.
2.7.8 The effect of volume exclusion by the nucleoid

In this section we analyze the effect of volume exclusion which might potentially be caused by a nucleoid if the DNA material acts as a diffusion barrier. Although it has been reported that the nucleoid does not necessarily influence protein diffusion \[SPL^{+14}\] it might still be possible that the diffusion of AtMinD is effected by interaction with the DNA material. For simplicity and as a reference for later studies, we here assume that the effect of this DNA material can be modeled as an effective excluded volume of different sizes.

We consider an elliptic diffusion barrier with the same proportions as the cell (see Fig. 2.8 A). This volume resides in the middle of the cell and is modeled as an obstacle with reflecting boundary conditions. We studied the impact of such an effective excluded volume for two different scenarios: when the \( \text{P}_{\text{NDP}} \) species shows finite recruitment (Fig. 2.8 B,C) and when the \( \text{P}_{\text{NTP}} \) species shows finite recruitment (Fig. 2.8 D,E). In both scenarios we changed the size of the elliptic volume as well as the nucleotide exchange rate which defines the typical length-scale of the pattern. We observe that as the excluded volume is increased the necessary minimal diffusion length \( l_{\lambda} \) to obtain a finite polarity shrinks. The reason becomes evident when we look at the local reaction volume defined by the diffusion length \( l_{\lambda} \) at mid cell vs. the equivalent reaction volume at the poles: If there is a diffusion barrier at a distance smaller than \( l_{\lambda} \) away from the membrane at mid-cell, the local reaction volume becomes smaller. If this reaction volume at midcell is smaller then the reaction volume at the poles the \( \text{P}_{\text{NDP}} \) particles start accumulating at mid-cell while the \( \text{P}_{\text{NTP}} \) species avoids mid-cell. However, if we decrease the diffusion length below the distance of the excluded volume from the membrane this reaction volume is no longer influenced by the volume exclusion effect and only dependent on the local curvature of the membrane. Thus, \( \text{P}_{\text{NDP}} \) is avoiding mid-cell while \( \text{P}_{\text{NTP}} \) is accumulating there. Depending on which pattern is amplified by the recruitment rates \( k_{dD} \) or \( k_{tT} \) the overall pattern resembles the pattern of the \( \text{P}_{\text{NDP}} \) respectively \( \text{P}_{\text{NTP}} \) species.

Taken together, for a small effective excluded volume (a geometry factor smaller than 40%) \( k_{dD} \) should be high in order to obtain accumulation of proteins on the poles, while for a large excluded volume (a geometry factor larger than 85%) \( k_{tT} \) should be higher. For intermediate sized proteins the diffusion length scale needs to be adapted such that the necessary local reaction volume ratios are obtained. This is possible by an adjustment of the nucleotide exchange rate \( \lambda \) or by changing the diffusion coefficient.

Here we considered the nucleoid as an effective static excluded volume with reflecting boundary conditions in two dimensions. The actual nucleoid is, however, a dynamic density distribution of DNA material in three dimensions \[FBW^{+13}\]. How the diffusion of AtMinD is effected by this is yet unknown to the best of our knowledge. Here we studied the possible case that it is acting as an effective diffusion barrier. However, we want to stress that the size of this diffusion barrier is
most likely not the same as the extension of the nucleoid since for low densities of DNA material the interaction might be negligible. It might even very well be that AtMinD is small enough such that its diffusion is not influenced by the nucleoid at all. Hence, it is currently not possible to make quantitative predictions about the effect of the nucleoid on the proposed pattern forming mechanism. Yet we can say that bipolarity can be obtained for a broad spectrum of possibilities.

Once more data about the interaction of the nucleoid with AtMinD becomes available a full study taking into account those parameters can be performed. Then it would be possible to make quantitative predictions about the influence of the nucleoid. In that case one needs to conduct the theoretical analysis in three dimensions since the ratio of membrane area to cytosolic volume changes in the full dimensional case. A nucleoid of the same dimensions as in a two dimensional study would take less volume and thus we would expect that its impact on the final pattern is smaller than in a two dimensional study.

The parameters can be found in table 2.1. The initial conditions where different than in the previous cases: we started with a homogeneous protein density of $500 \ast 4/10.5\mu m^{-1}$ on the membrane which is approximately equivalent to a homogeneous protein density of $500\mu m^{-2}$ in the cytosol in the absence of an excluding volume. The reasoning behind this choice is to keep the amount of particles constant in the cell when its inner volume becomes reduced due to the volume exclusion. The simulation time was set to 150ks to ensure that it reaches equilibrium.

### 2.7.9 Caption figure 2.8

**Volume exclusion by the nucleoid.** (A) We model effective volume exclusion by the nucleoid by placing an ellipse with reflecting boundary conditions in the middle of the cell. This inner ellipse has the same proportions as the cell: the axes of the inner axis are obtained by multiplying the axis of the cell by a geometry factor. We show here three examples where this geometry factor is 0.3, 0.6, and 0.95, respectively. (B) Three examples for the pattern of the total protein density (green), the density of $P_{NDP}$ (red) and of $P_{NTP}$ (blue) for different sizes of the excluded volume and different values of the diffusion length scale (values marked by grey crosses in C)). The recruitment rate of $P_{NDP}$ is $k_{ND}=0.5\mu m/s^2$ while the recruitment of $P_{NTP}$ is switched off. The polarity here is significantly smaller than one. The upper plot corresponds to a large excluded volume (geometry factor=0.95) and a small diffusion length scale ($l_\lambda=\sqrt{(16\mu m^2/s)/213s^{-1}} \approx 0.27\mu m$). In this plot the proteins accumulate at mid-cell. The middle plot shows the results for an intermediate excluded volume (geometry factor=0.6) and a small diffusion length scale ($l_\lambda=\sqrt{(16\mu m^2/s)/213s^{-1}} \approx 0.27\mu m$). In this plot the protein density peaks at the poles as well as at mid-cell. Thus the effective polarity is close to one. The lower plot shows the results for a small excluded volume (geometry factor=0.3) and a larger diffusion length scale ($l_\lambda=\sqrt{(16\mu m^2/s)/48s^{-1}} \approx 0.58\mu m$). In this
plot the protein density peaks strongly at the cell poles. Thus the polarity is significantly larger than one. (C) For P_{NDP} being the species with the higher recruitment dominance, small excluded volume leads to the accumulation of the proteins at the poles (polarity larger than 1) while large excluded volume leads to accumulation at mid-cell (polarity smaller than 1). For intermediate excluded volumes both phases are possible depending on the magnitude of the diffusion length $l_\lambda$. Small length scales lead to a polarity larger than one while larger length scales lead to a polarity smaller than one. For $l_\lambda$ large enough, the polarity becomes equal to 1. The recruitment rate of P_{NDP} is $k_{dD} = 0.5 \mu m/s^2$ while the recruitment of P_{NTP} is switched off. For visualization purposes the color bar for the polarity is clipped at a value of 1.5 from above and at a value of 0.5 from below (same holds for D)). The grey crosses show the parameter values for the three examples in B). The sweep was performed for 50 logarithmically spaced values of the geometry factor between 0.2953 and 0.9502, and for 46 logarithmically spaced values of $\lambda$ between 2.7s$^{-1}$ and 671.9s$^{-1}$ (same holds for D)). (D) If P_{NTP} is the species with the higher recruitment dominance the behavior for different sizes of excluded volume is inverted with respect to (C). Here accumulation at the poles is obtained for large excluded volumes while small excluded volumes lead to accumulation at mid-cell. For intermediate value the polarity is close to one thus neither the poles nor mid-cell is preferred. As for the case with $k_{dD}>0$ large values of $l_\lambda$ lead to a polarity close to one. The recruitment rate of P_{NTP} is $k_{tT} = 0.5 \mu m/s^2$ while the recruitment of P_{NDP} is switched off. The grey crosses show the parameter values for the three examples in E). (E) Similar to B) three examples of the protein pattern are shown but now for $k_{dD}=0$ and $k_{tT}=0.5 \mu m/s^2$. The parameter values of $\lambda$ and the geometry factor correspond to the plots in B): The upper plot shows that the proteins accumulate at the poles for a large excluded volume. The pattern originates from the spatial P_{NTP} distribution, whereas the P_{NDP} species shows no significant pattern. The polarity in this case is larger than one. The center plot shows a depletion of the proteins at mid-cell and the poles for an intermediate excluded volume. The polarity is close to one. The lower plot shows strong depletion of proteins at the poles (but not at mid-cell) and thus a polarity smaller than one.
Figure 2.6: Overall protein density in the cytosol. (A) Overall cytosolic protein density, \( u_c = u_D + u_T \), for the same parameters as in Fig.2.2F where recruitment is switched on. The protein density is enhanced at mid-cell and depleted at the poles. (B) Overall cytosolic protein density, \( u_c = u_D + u_T \), for the same parameters as in Fig.2.2C, where recruitment is switched off but the attachment rates are unbalanced with \( P_{NTP} \) having the higher membrane affinity: \( \omega_D^+ = 0.1 \mu m/s \) and \( \omega_T^+ = 1 \mu m/s \). The protein density exhibits maxima at both cell poles and a minimum at mid-cell. The profile is inhomogeneous but much less pronounced than in panel A. (C) Overall cytosolic protein density, \( u_c = u_D + u_T \), for the same parameters as in Fig.2.2C, where recruitment is switched off but the attachment rates are unbalanced with \( P_{NDP} \) having the higher membrane affinity: \( \omega_D^+ = 1 \mu m/s \) and \( \omega_T^+ = 0.1 \mu m/s \). The protein density exhibits maxima at mid-cell and minima at both cell-poles. The profile is inhomogeneous but much less pronounced than in panel A.
2. Geometry induced protein pattern formation

Figure 2.7: A hopping process on a one-dimensional grid with bulk nodes $x$ at a distance $a$ (open circles), and with a membrane node on the left (square). In the continuum limit $a \to 0$, one obtains diffusion in the bulk which is coupled to the membrane by a reactive boundary conditions which balances diffusive flux on and off the membrane with the attachment and detachment processes.
2.7. Supplement: Geometry induced protein pattern formation

Figure 2.8: Caption see section 2.7.9
Chapter 3

Learning universal computations with spikes

3.1 Abstract

Providing the neurobiological basis of information processing in higher animals, spiking neural networks must be able to learn a variety of complicated computations, including the generation of appropriate, possibly delayed reactions to inputs and the self-sustained generation of complex activity patterns, e.g. for locomotion. Many such computations require previous building of intrinsic world models. Here we show how spiking neural networks may solve these different tasks. Firstly, we derive constraints under which classes of spiking neural networks lend themselves to substrates of powerful general purpose computing. The networks contain dendritic or synaptic nonlinearities and have a constrained connectivity. We then combine such networks with learning rules for outputs or recurrent connections. We show that this allows to learn even difficult benchmark tasks such as the self-sustained generation of desired low-dimensional chaotic dynamics or memory-dependent computations. Furthermore, we show how spiking networks can build models of external world systems and use the acquired knowledge to control them. ¹

3.2 Introduction

The understanding of neural network dynamics on the mesoscopic level of hundreds and thousands of neurons and their ability to learn highly complicated computations

¹This chapter is based on Dominik Thalmeier, Marvin Uhlmann, Hilbert J Kappen, Raoul-Martin Memmesheimer. Learning universal computations with spikes PLOS Comput Biol 12 (6), e1004895, 2016.
is a fundamental open challenge in neuroscience. For biological systems, such an understanding will allow to connect the microscopic level of single neurons and the macroscopic level of cognition and behavior. In artificial computing, it may allow to propose new, possibly more efficient computing schemes.

Randomly connected mesoscopic networks can be a suitable substrate for computations [MNM02, JH04, MJS07, SA09, SB13], as they reflect the input in a complicated, nonlinear way and at the same time maintain, like a computational “reservoir”, fading memory of past inputs as well as of transformations and combinations of them. This includes the results of computations on current and past inputs. Simple readout neurons may then learn to extract the desired result; the computations are executed in real time, i.e. without the need to wait for convergence to an attractor (“reservoir computing”) [MNM02, JH04]. Non-random and adaptive network connectivity can change performance [JL09, LPT09, KM13].

Networks with higher computational power, in particular with the additional ability to learn self-sustained patterns of activity and persistent memory, require an output feedback or equivalent learning of their recurrent connections [JH04, MJS07]. However, network modeling approaches achieving such universal (i.e. general purpose) computational capabilities so far concentrated on networks of continuous rate units [JH04, SA09], which do not take into account the characteristics that neurons in biological neural networks communicate via spikes. Indeed, the dynamics of spiking neural networks are discontinuous, usually highly chaotic, variable, and noisy. Readouts of such spiking networks show low signal-to-noise ratios. This hinders computations following the described principle in particular in presence of feedback or equivalent plastic recurrent connections, and has questioned it as model for computations in biological neural systems [JM05, MG05, WML13].

Here we first introduce a class of recurrent spiking neural networks that are suited as a substrate to learn universal computations. They are based on standard, established neuron models, take into account synaptic or dendritic nonlinearities and are required to respect some structural constraints regarding the connectivity of the network. To derive them we employ a precise spike coding scheme similar to ref. [BMD13], which was introduced to approximate linear continuous dynamics.

Thereafter we endow the introduced spiking networks with learning rules for either the output or the recurrent connection weights and show that this enables them to learn equally complicated, memory dependent computations as non-spiking continuous rate networks. The spiking networks we are using have only medium sizes, between tens and a few thousands of neurons, like networks of rate neurons employed for similar tasks. We demonstrate the capabilities of our networks by applying them to challenging learning problems which are of importance in biological contexts. In particular, we show how spiking neural networks can learn the self-sustained generation of complicated dynamical patterns, and how they can build world models, which allow to compute optimal actions to appropriately influence an environment.
3.3 Results

3.3.1 Continuous signal coding spiking neural networks (CSNs)

Network architecture

For our study, we use leaky integrate-and-fire neurons. These incorporate crucial features of biological neurons, such as operation in continuous time, spike generation and reset, while also maintaining some degree of analytical tractability. A network consists of $N$ neurons. The state of a neuron $n$ is given by its membrane potential $V_n(t)$. The membrane potential performs a leaky integration of the input and a spike is generated when $V_n(t)$ reaches a threshold, resulting in a spiketrain

$$s_n(t) = \sum_{t_n} \delta(t - t_n)$$

(3.1)

with spike times $t_n$ and the Dirac delta-distribution $\delta$. After a spike, the neuron is reset to the reset potential, which lies $\theta$ below the threshold. The spike train generates a train of exponentially decaying normalized synaptic currents

$$r_n(t) = \sum_{t_n} e^{-\lambda_s(t-t_n)} \Theta(t - t_n) \iff \dot{r}_n(t) = -\lambda_s r_n(t) + s_n(t),$$

(3.2)

where $\tau_s = \lambda_s^{-1}$ is the time constant of the synaptic decay and $\Theta(.)$ is the Heaviside theta-function.

Throughout the article we consider two closely related types of neurons, neurons with saturating synapses and neurons with nonlinear dendrites (cf. Fig. 3.1). In the model with saturating synapses (Fig. 3.1a), the membrane potential $V_n(t)$ of neuron $n$ obeys

$$\dot{V}_n(t) = -\lambda_V V_n(t) + \sum_{m=1}^{N} A_{nm} \tanh(\gamma r_m(t)) + V_r \lambda_s r_n(t)$$

$$- \theta s_n(t) + I_{e,n}(t),$$

(3.3)

with membrane time constant $\tau_m = \lambda_V^{-1}$. 
3. Learning universal computations with spikes
Figure 3.1 (previous page): Coding of continuous signals in neurons with saturating synapses (a,b) and nonlinear dendrites (c,d). (a,b): A neuron with saturating synapses (a) that directly codes for a continuous signal (b). Panel (a) displays the neuron with an axon (red) and dendrites (dark blue) that receive inputs from the axons of other neurons (axons at the bottom) via saturating synapses (symbolized by sigmoids at the synaptic contacts). The currents entering the soma are weighted sums of input spike trains that are synaptically filtered (generating scaled normalized synaptic currents $\gamma r_n(t)$, synaptic time scale $\tau_s$) and thereafter subject to a saturating synaptic nonlinearity. External inputs (axons at the top) are received without saturation. The continuous signal $x(t)$ (panel b left hand side, green) is the sum of the neuron’s membrane potential $V(t)$ (red) and its scaled normalized synaptic current $\theta r(t)$ (dark blue). $r(t)$ is a low-pass filtered version of the neuron’s spike train $s(t)$ (light blue in red box). If $x(t) > 0$, the time scale of $x(t)$ should be large against the synaptic time scale $\tau_s$ and $x(t)$ should predominantly be large against the neuron’s threshold, $\theta/2$ (panel b right hand side, assumptions [1,2] in the main text). $x(t)$ is then already well approximated by $\theta r(t)$, while $V(t)$ is oscillating between $\pm \theta/2$. If $x(t) \leq 0$, we have $V(t) \leq 0$, no spikes are generated and $r(t)$ quickly decays to zero, such that we predominantly have $r(t) \approx 0$ and $x(t)$ is well approximated by $V(t)$ (cf. Equation (3.9)).

(c,d): Two neurons with nonlinear dendrites (c) from a larger network that distributes codes for a continuous signal (d). (c): Each neuron has an axon (red) and different types of dendrites (cyan, light blue and dark blue) that receive inputs from the axons of other neurons (axons at the bottom) via fast or slow conventional synapses (highlighted by circles and squares). Linear dendrites with slow synapses (cyan with circle contacts) generate somatic currents that are weighted linear sums of low-pass filtered presynaptic spike trains (weighted sums of the $r_n(t)$). Linear dendrites with fast synapses (light blue with square contacts) generate somatic currents with negligible filtering (weighted sums of the spike trains $s_n(t)$). Spikes arriving at a nonlinear dendrite (dark blue) are also filtered (circular contact). The resulting $r_n(t)$ are weighted, summed up linearly in the dendrite and subjected to a saturating dendritic nonlinearity (symbolized by sigmoids at dendrites), before entering the soma. We assume that the neurons have nonlinear dendrites that are located in similar tissue areas, such that they connect to the same sets of axons and receive similar inputs. (d): All neurons in the network together encode $J$ continuous signals $x(t)$ (one displayed in green) by a weighted sum of their membrane potentials $V(t)$ (two traces of different neurons displayed in red) and their normalized PSCs $r(t)$ (two traces displayed in cyan). The $\Gamma r(t)$ alone already approximate $x(t)$ well. The neurons’ output spike trains $s(t)$ (light blue in red box) generate slow and fast inputs to other neurons. (Note that spikes can be generated due to suprathreshold excitation by fast inputs. Since we plot $V(t)$ after fast inputs and possible resets, the corresponding threshold crossings do not appear.)
3. Learning universal computations with spikes

The saturation of synapses, e.g. due to receptor saturation or finite reversal potentials, acts as a nonlinear transfer function [BFR04, DA01], which we model as a tanh-nonlinearity (since \( r_m(t) \geq 0 \) only the positive part of the \( \tanh \) becomes effective). We note that this may also be interpreted as a simple implementation of synaptic depression: A spike generated by neuron \( m \) at \( t_m \) leads to an increase of \( r_m(t_m) \) by 1. As long as the synapse connecting neuron \( m \) to neuron \( n \) is far from saturation (linear part of the \( \tanh \)-function) this leads to the consumption of a fraction \( \gamma \) of the synaptic “resources” and the effect of the spike on the neuron is approximately the effect of a current \( A_{nm} \gamma e^{-\lambda_s(t-t_m)} \Theta(t-t_m) \). When a larger number of such spikes arrive in short time such that the consumed resources accumulate to 1 and beyond, the synapse saturates at its maximum strength \( A_{nm} \) and the effect of individual inputs is much smaller than before. The recovery from depression is here comparably fast, it takes place on a timescale of \( \lambda_s^{-1} \) (compare, e.g., [AR04]).

The reset of the neuron is incorporated by the term \( -\theta s_n(t) \). The voltage lost due to this reset is partially recovered by a slow recovery current (afterdepolarization) \( V_r \lambda_s r_n(t) \); its temporally integrated size is given by the parameter \( V_r \). This is a feature of many neurons e.g. in the neocortex, in the hippocampus and in the cerebellum [Bea07], and may be caused by different types of somatic or dendritic currents, such as persistent and resurgent sodium and calcium currents, or by excitatory autapses [LMFS96, Bek09]. It provides a simple mechanism to sustain (fast) spiking and generate bursts, e.g. in response to pulses. \( I_{e,n}(t) \) is an external input, its constant part may be interpreted as sampling slow inputs specifying the resting potential that the neuron asymptotically assumes for long times without any recurrent network input. We assume that the resting potential is halfway between the reset potential \( V_{res} \) and the threshold \( V_{res} + \theta \). We set it to zero such that the neuron spikes when the membrane potential reaches \( \theta/2 \) and resets to \( -\theta/2 \). To test the robustness of the dynamics we sometimes add a white noise input \( \eta_n(t) \) satisfying \( \langle \eta_n(t) \eta_m(t') \rangle = \sigma^2 \delta_{nm} \delta(t-t') \) with the Kronecker delta \( \delta_{nm} \).

For simplicity, we take the parameters \( \lambda_V, \theta, V_r \) and \( \gamma, \lambda_s \) identical for all neurons and synapses, respectively. We take the membrane potential \( V_n \) and the parameters \( V_r \) and \( \theta \) dimensionless, they can be fit to the voltage scale of biological neurons by rescaling with an additive and a multiplicative dimensionful constant. Time is measured in seconds.

We find that networks of the form Equation (3.3) generate dynamics suitable for universal computation similar to continuous rate networks [JH04, SA09], if \( 0 < \lambda_x \ll \lambda_s \), where \( \lambda_x = \lambda_s \left( 1 - \frac{V}{\theta} \right) \), \( A_{nm} \) sufficiently large and \( \gamma \) small. The conditions result from requiring the network to approximate a nonlinear continuous dynamical system (see next section).

An alternative interpretation of the introduced nonlinearity is that the neurons have nonlinear dendrites, where each nonlinear compartment is small such that it receives at most one (conventional, nonsaturating) synapse. \( A_{nm} \) is then the strength of the coupling from a dendritic compartment to the soma. This interpretation
3.3. Results

suggests an extension of the neuron model allowing for several dendrites per neuron, where the inputs are linearly summed up and then subjected to a saturating
dendritic nonlinearity [LH05, Mem10, CHG13]. Like the previous model, we find
that such a model has to satisfy additional constraints to be suitable for universal
computation:

Neurons with nonlinear dendrites need additional slow and fast synaptic contacts
which arrive near the soma and are summed linearly there (Fig. 3.1c). Such
structuring has been found in biological neural networks [BH11]. We gather the
different components into a dynamical equation for $V_n$ as

\[
\dot{V}_n(t) = -\lambda_V V_n(t) + \sum_{j=1}^{J} D_{nj} \tanh \left( \sum_{m=1}^{N} W_{njm} r_m(t) \right) + \sum_{m=1}^{N} \tilde{U}_{nm} r_m(t) - \sum_{m=1}^{N} U_{nm} s_m(t) + \sum_{j=1}^{J} \Gamma_{jn} I_{e,j}(t).
\]

(3.4)

$D_{nj}$ is the coupling from the $j$th dendrite of neuron $n$ to its soma. The total number
of dendrites and neurons is referred to as $J$ and $N$ respectively. $W_{njm}$ is the coupling
strength from neuron $m$ to the $j$th nonlinear dendrite of neuron $n$. The slow,
significantly temporally filtered inputs from neuron $m$ to the soma of neuron $n$, $\tilde{U}_{nm} r_m(t)$, have connection strengths $\tilde{U}_{nm}$. The fast ones, $U_{nm} s_m(t)$, have negligible
synaptic filtering (i.e. negligible synaptic rise and decay times) as well as negligible
conduction delays. The resets and recoveries are incorporated as diagonal elements
of the matrices $U_{nm}$ and $\tilde{U}_{nm}$. To test the robustness of the dynamics, also here
we sometimes add a white noise input $\eta_n(t)$. To increase the richness of the
recurrent dynamics and the computational power of the network (cf. [HZS06] for
disconnected units without output feedback) we added inhomogeneity, e.g. through
the external input current in some tasks. In the control/mental exploration task, we
added a constant bias term $b_j$ as argument of the tanh to introduce inhomogeneity.

We find that the network couplings $D, W, U$ and $\tilde{U}$ (3.4) (we use bold letters
for vectors and matrices) should satisfy certain interrelations. As motivated in
the subsequent section and derived in the supporting material, their components
may be expressed in terms of the components of a $J \times N$ matrix $\Gamma$, and a $J \times J$
matrix $\Lambda$ as $D_{nj} = \sum_{i=1}^{J} \Gamma_{ni} \Lambda_{ij}$, $W_{njm} = \Gamma_{jm}$, $\tilde{U}_{nm} = a \sum_{j=1}^{J} \Gamma_{jn} \Gamma_{jm} + \mu \lambda_x \delta_{nm}$, $U_{nm} = \sum_{j=1}^{J} \Gamma_{jn} \Gamma_{jm} + \mu \delta_{nm}$, where $a = \lambda_x - \lambda_s$ and $\mu \geq 0$ is small (see also Table 3.1 for
an overview). The thresholds are chosen identical, $\theta^n = \theta$, see Methods.

Again, the conditions result from requiring the network to approximate a non-
linear continuous dynamical system. This system, Equation (3.11), is characterized
by the $J \times J$ coupling matrix $\Lambda$ and a $J$-dimensional input $c(t)$ whose components
3. Learning universal computations with spikes

<table>
<thead>
<tr>
<th>explanation</th>
<th>optimal value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{nj}$</td>
<td>coupling from the $j$th dendrite of neuron $n$ to its soma</td>
</tr>
<tr>
<td>$W_{njm}$</td>
<td>coupling strength from neuron $m$ to the $j$th nonlinear dendrite of neuron $n$</td>
</tr>
<tr>
<td>$\tilde{U}_{nm}$</td>
<td>slow coupling from neuron $m$ to neuron $n$; diagonal elements incorporate a recovery current</td>
</tr>
<tr>
<td>$U_{nm}$</td>
<td>fast coupling from neuron $m$ to neuron $n$; diagonal elements incorporate the reset</td>
</tr>
<tr>
<td>$\theta^n$</td>
<td>threshold of neuron $n$</td>
</tr>
</tbody>
</table>

Table 3.1: Parameters of a network of neurons with nonlinear dendrites (cf. Equation (3.4)) and their optimal values.

are identical to the $J$ independent components of the external input current $I_e$ in equation (3.4); the matrix $\Gamma$ is a decoding matrix that fixes the relation between spiking and continuous dynamics (see next section). We note that the matrices $\Gamma$ and $A$ are largely unconstrained, such that the coupling strengths maintain a large degree of arbitrariness. Ideally, $W_{njm}$ is independent of $n$, therefore neurons have dendrites that are similar in their input characteristics to dendrites in some other neurons (note that $D$ may have zero entries, so dendrites can be absent). We interpret these as dendrites that are located in a similar tissue area and therefore connect to the same axons and receive similar inputs (cf. Fig. 3.1c for an illustration). The interrelations between the coupling matrices might be realized by spike-timing dependent synaptic or structural plasticity. Indeed, for a simpler model and task, appropriate biologically plausible learning rules have been recently highlighted [BBMD12,BD15]. We tested robustness of our schemes against structural perturbations (see Figs. 3.8, 3.9 in the supplement 3.6), in particular for deviations from the $n$-independence of $W_{njm}$ (Fig. 3.8 in the supplement 3.6).

The networks Equation (3.3) with saturating synapses have a largely unconstrained topology, in particular they can satisfy the rule that neurons usually act only excitatorily or inhibitorily. For the networks Equation (3.4) with nonlinear dendrites, it is less obvious how to reconcile the rule with the constraints on the network connectivity. Solutions for this have been suggested in simpler systems and are subject to current research [BMD13].

The key property of the introduced neural architecture is that the spike trains generated by the neurons encode with high signal-to-noise ratio a continuous signal that can be understood in terms of ordinary differential equations. In the following section we show how this signal is decoded from the spike trains. Thereafter, we may conclude that the spiking dynamics are sufficiently “tamed” such that standard
3.3. Results

Learning rules can be applied to learn complicated computations.

**Direct encoding of continuous dynamics**

The dynamics of a neural network with \( N \) integrate-and-fire neurons consist of two components, the sub-threshold dynamics \( \mathbf{V}(t) = (V_1(t), \ldots, V_N(t))^T \) of the membrane potentials and the spike trains \( \mathbf{s}(t) = (s_1(t), \ldots, s_N(t))^T \) (Equation 3.1), which are temporal sequences of \( \delta \)-distributions. In the model with saturating synapses, all synaptic interactions are assumed to be significantly temporally filtered, such that the \( V_n(t) \) are continuous except at reset times after spiking (Equation (3.3)).

We posit that the \( \mathbf{V}(t) \) and the \( \mathbf{s}(t) \) should together form some \( N \)-dimensional continuous dynamics \( \mathbf{x}(t) = (x_1(t), \ldots, x_N(t))^T \). The simplest approach is to setup \( \mathbf{x}(t) \) as a linear combination of the two components \( \mathbf{V}(t) \) and \( \mathbf{s}(t) \). To avoid infinities in \( x_n(t) \), we need to eliminate the occurring \( \delta \)-distributions by employing a smoothed version of \( s_n(t) \). This should have a finite discontinuity at spike times such that the discontinuity in \( V_n(t) \) can be balanced. A straightforward choice is to use \( \theta r_n(t) \) (Equation (3.2)) and to set

\[
V_n(t) + \theta r_n(t) = x_n(t). \tag{3.5}
\]

(cf. Fig. 3.1b). When the abovementioned conditions on \( \lambda_x, \lambda_s, A \) and \( \gamma \) are satisfied (cf. end of the section introducing networks with saturating synapses), the continuous signal \( \mathbf{x}(t) \) follows a system of first order nonlinear ordinary differential equations similar to those describing standard non-spiking continuous rate networks used for computations (cf. \[JH04,SA09,LJS12]\] and Equation (3.11) below),

\[
\dot{x}_n(t) = -\lambda_V \ [x_n(t)]_- - \lambda_x \ [x_n(t)]_+ + \sum_{m=1}^{N} A_{nm} \tanh \left( \frac{\gamma}{\theta} \ [x_m(t)]_+ \right) + I_{e,n}(t), \tag{3.6}
\]

with the rectifications \( [x_n(t)]_+ = \max(x_n(t), 0) \), \( [x_n(t)]_- = \min(x_n(t), 0) \). We call spiking networks where this is the case continuous signal coding spiking neural networks (CSNs).

Except for the rectifications, Equation (3.6) has a standard form for non-spiking continuous rate networks, used for computations \[JH04,SA09,LJS12]\]. A salient choice for \( \lambda_x \) is \( \lambda_x = \lambda_V \), i.e. \( V_r = \left( 1 - \frac{\lambda_s}{\lambda_x} \right) \theta \), such that the rectifications outside the \( \tanh \)-nonlinearity vanish. Equation (3.6) generates dynamics that are different from the standard ones in the respect that the trajectories of individual neurons are, e.g. for random Gaussian matrices \( A \), not centered at zero. However, they can satisfy the conditions for universal computation (enslaveability/echo state property and high dimensional nonlinear dynamics) and generate longer-term fading memory for appropriate scaling of \( A \). Also the corresponding spiking networks are then suitable for fading memory-dependent computations. Like for the standard networks \[Jae01,YJK12]\], we can derive sufficient conditions to guarantee that the dynamics
3. Learning universal computations with spikes

Equation (3.6) are enslaveable by external signals (echo state property). \( \|A\| < \min(\lambda_V, \lambda_x) \), where \( \|A\| \) is the largest singular value of the matrix \( A \), provides such a condition (see Supplementary material for the proof). The condition is rather strict, our applications indicate that the CSNs are also suited as computational reservoirs when it is violated. This is similar to the situation in standard rate network models \([\text{Jae}01]\). We note that if the system is enslaved by an external signal, the time scale of \( x_n(t) \) is largely determined by this signal and not anymore by the intrinsic scales of the dynamical system.

We will now show that spiking neural networks Equation (3.3) can encode continuous dynamics Equation (3.6). For this we derive the dynamical equation of the membrane potential (3.3) from the dynamics of \( x(t) \) using the coding rule Equation (3.5), the dynamical equation (3.2) for \( r_n(t) \) and the rule that a spike is generated whenever \( V_n(t) \) reaches threshold \( \theta/2 \): We first differentiate Equation (3.5) to eliminate \( \dot{x}_n(t) \) from Equation (3.6) and employ Equation (3.2) to eliminate \( \dot{r}_n(t) \). The resulting expression for \( \dot{V}_n(t) \) reads

\[
\dot{V}_n(t) = -\lambda_V [x_n(t)]_+ - \lambda_x [x_n(t)]_- + \sum_{n=1}^{N} A_{nn} \tanh\left(\frac{\gamma}{\theta} [x_n(t)]_+ \right) - \theta s_n(t) + \lambda_x \theta r_n(t) + I_{e,n}(t). \tag{3.7}
\]

It already incorporates the resets of size \( \theta \) (cf. the term \( -\theta s_n(t) \)), they arise since \( x_n(t) = V_n(t) + \theta r_n(t) \) is continuous and \( r_n(t) \) increases by one at spike times (thus \( V \) must decrease by \( \theta \)). We now eliminate the occurrences of \( [x_n(t)]_+ \) and \( [x_n(t)]_- \).

For this, we make two assumptions (cf. Fig. 3.1b) on the \( x_n(t) \) if they are positive:

[1] The dynamics of \( x_n(t) \) are slow against the synaptic timescale \( \tau_s \),

[2] the \( x_n(t) \) assume predominantly values \( x_n(t) \gg \theta/2 \).

First we consider the case \( x_n(t) > 0 \). Since \( V_n(t) \) is reset when it reaches its threshold value \( \theta/2 \), \( V_n(t) \) is always smaller than \( \theta/2 \). Thus, given \( V_n(t) > 0 \) assumption [2] implies that we can approximate \( x_n(t) \approx \theta r_n(t) \), as the contribution of \( V_n(t) \) is negligible because \( V_n(t) \leq \theta/2 \). This still holds if \( V_n(t) \) is negative and its absolute value is not large against \( \theta/2 \). Furthermore, assumption [1] implies that smaller negative \( V_n(t) \) cannot co-occur with positive \( x_n(t) \): \( r_n(t) \) is positive and in the absence of spikes it decays to zero on the synaptic time scale \( \tau_s \) (Equation (3.2)). When \( V_n(t) < 0 \), neuron \( n \) is not spiking anymore. Thus when \( V_n(t) \) is shrinking towards small negative values and \( r_n(t) \) is decaying on a timescale of \( \tau_s \), \( x_n(t) \) is also decaying on a time-scale \( \tau_s \). This contradicts assumption [1]. Thus when \( x_n(t) > 0 \), the absolute magnitude of \( V_n(t) \) is on the order of \( \theta/2 \). With assumption [2] we can thus set \( x_n(t) \approx \theta r_n(t) \), whenever \( x_n(t) > 0 \), neglecting contributions of size \( \theta/2 \).

Now we consider \( x_n(t) \leq 0 \). This implies \( V_n(t) \leq 0 \) (since always \( r_n(t) \geq 0 \)) as well as a quick decay of \( r_n(t) \) to zero. When \( x_n(t) \) assumes values significantly
below zero, assumption [1] implies that we have $x_n(t) \approx V_n(t)$ and $r_n(t) \approx 0$, otherwise $x_n(t)$ must have changed from larger positive (assumption [2]) to larger negative values on a timescale of $\tau_s$.

The approximate expressions may be gathered in the replacements $[x_n(t)]_+ = \theta r_n(t)$ and $[x_n(t)]_- = [V_n(t)]_-$. Using these in Equation (3.7) yields together with $\lambda_x = \lambda_s \left(1 - \frac{\theta}{\lambda} \right)$

$$\dot{V}_n(t) = -\lambda V_n(t) + \sum_{m=1}^{N} A_{nm} \tanh(\gamma r_m(t)) + V_i \lambda_s r_n(t) - \theta s(t) + I_{e,n}(t). \quad (3.8)$$

Note that our replacements allowed to eliminate the biologically implausible $V$-dependencies in the interaction term.

To simplify the remaining $V_n(t)$-dependence, we additionally assume that

[2'] $x_n(t)$ assumes predominantly values $x_n(t) \gg \lambda V \theta / (2 \lambda_x)$,

if $x_n(t)$ is positive. This can be stricter than [2] depending on the values of $\lambda_x$ and $\lambda_V$. For positive $x_n(t)$, where $\lambda V [x_n(t)]_-$ in Equation (3.7) is zero, $\lambda V V_n(t)$ has an absolute magnitude on the order of $\lambda V \theta / 2$ (see the arguments above). [2'] implies that this is negligible against $-\lambda_x [x_n(t)]_+$. For negative $x_n(t)$, we still have $x_n(t) \approx V_n(t)$. This means that we may replace $-\lambda V [x_n(t)]_-$ by $\lambda V V_n(t)$ in Equation (3.7). Taken together, under the assumptions [1,2,2'] we may use the replacements

$$[x_n(t)]_+ \approx \theta r_n(t)$$

$$[x_n(t)]_- \approx V_n(t) \quad (3.9)$$

in Equation (3.7), which directly yield Equation (3.3). Note that this also implies $r_n(t) \gg \theta / 2$ if the neuron is spiking, so during active periods inter-spike-intervals need to be considerably smaller than the synaptic time scale.

Equation (3.6) implies that the assumptions are justified for suitable parameters: For fixed parameters $\tau_s = \lambda_s^{-1}$ and $\theta$ of the $r$-dynamics, we can choose sufficiently small $\lambda_x$, large $A_{nm}$, and small $\gamma$ to ensure assumptions [1,2,2'] (cf. the conditions highlighted in the section “Network architecture”). On the other hand, for given dynamics Equation (3.6), we can always find a spiking system which generates the dynamics via Equations (3.3), (3.2) and (3.5), and satisfies the assumptions: We only need to choose $\tau_s$ sufficiently small such that [1] is satisfied and the spike threshold sufficiently small such that [2,2'] are satisfied. For the latter, $\gamma$ needs to be scaled like $\theta$ to maintain the dynamics of $x_n$ and $V_r$ needs to be computed from the expression for $\lambda_x$. Interestingly, we find that also outside the range where the assumptions are satisfied, our approaches can still generate good results.

The recovery current in our model has the same time constant as the slow synaptic current. Indeed, experiments indicate that they possess the same characteristic
3. Learning universal computations with spikes

Timescales: Timescales for NMDA [WSWG08] and slow GABA\textsubscript{A} [PGKP07, SM08] receptor mediated currents are several tens of milliseconds. Afterdepolarizations have timescales of several tens of milliseconds as well [Sto87, RB97, Bea07, CY08, BR09]. Another prominent class of slow inhibitory currents is mediated by GABA\textsubscript{B} receptors and has timescales of one hundred to a few hundreds of milliseconds [LJS'97].

We remark that in our model the time constants of the afterdepolarization and the synaptic input currents may also be different without changing the dynamics: Assume that the synaptic time constant is different from that of the recovery current, but still satisfies the conditions that it is large against the inter-spike-intervals when the neuron is spiking and small against the timescale of $[x_n(t)]_+$. The synaptic current generated by the spike train of neuron $n$ will then be approximately continuous and the filtering does not seriously affect its overall shape beyond smoothing out the spikes. As a consequence, the synaptic and the recovery currents are approximately proportional up to a constant factor that results from the different integrated contribution of individual spikes to them. Rescaling $\gamma$ by this factor thus yields dynamics equivalent to the one with identical time constants.

Distributed encoding of continuous dynamics

In the above-described simple CSNs (CSNs with saturating synapses), each spiking neuron gives rise to one nonlinear continuous variable. The resulting condition that the inter-spike-intervals are small against the synaptic time constants if the neuron is spiking may in biological neural networks be satisfied for bursting or fast spiking neurons with slow synaptic currents. It will be invalid for different neurons and synaptic currents. The condition becomes unnecessary when the spiking neurons encode continuous variables collectively, i.e. if we partially replace the temporal averaging in $r_n(t)$ by an ensemble averaging. This can be realized by an extension of the above model, where only a lower, say $J-$, dimensional combination $x(t)$ of the $N-$dimensional vectors $V(t)$ and $r(t)$ is continuous,

$$x(t) = LV(t) + \tilde{\Gamma}r(t), \quad (3.10)$$

where $L$ and $\tilde{\Gamma}$ are $J \times N$ matrices (note that Equation (3.5) is a special case with $N = J$ and diagonal matrices $L$ and $\tilde{\Gamma}$). We find that spiking networks with nonlinear dendrites Equation (3.4) can encode such a lower dimensional variable $x(t)$. The $x(t)$ satisfy $J$-dimensional standard equations describing non-spiking continuous rate networks used for reservoir computing [JH04, SA09, LJS12],

$$\dot{x}(t) = -\lambda_x x(t) + A \tanh(x(t)) + c(t). \quad (3.11)$$

We denote the resulting spiking networks as CSNs with nonlinear dendrites.

The derivation (see Supplementary material for details) generalizes the ideas introduced in refs. [BD11, BBMD12, BMD13] to the approximation of nonlinear
3.3. Results

dynamical systems: We assume an approximate decoding equation (cf. also Equation (3.9)),

\[ x(t) \approx \Gamma r(t), \]  

(3.12)

where \( \Gamma \) is a \( J \times N \) decoding matrix and employ an optimization scheme that minimizes the decoding error resulting from Equation (3.12) at each time point. This yields the condition that a spike should be generated when a linear combination of \( x(t) \) and \( r(t) \) exceeds some constant value. We interpret this linear combination as membrane potential \( V(t) \). Solving for \( x(t) \) gives \( L \) and \( \tilde{\Gamma} \) in terms of \( \Gamma \) in Equation (3.10). Taking the temporal derivative yields \( \dot{V}(t) \), first in terms of \( \dot{x}(t) \) and \( \dot{r}(t) \) and after replacing them via Equations (3.2), (3.11), in terms of \( x(t) \), \( r(t) \) and \( s(t) \). We then eliminate \( x(t) \) using (3.12) and add a membrane potential leak term for biological realism and increased stability of numerical simulations. This yields Equation (3.4) together with the optimal values of the parameters given in Table 3.1. We note that the difference to the derivation in ref. [BMD13] is the use of a nonlinear equation when replacing \( \dot{x}(t) \). We further note that the spiking approximation of the continuous dynamics becomes exact, if in the last step \( x(t) \) is eliminated using Equation (3.10) and the leak term is omitted as it does not arise from the formalism in contrast to the case of CSNs with saturating synapses. Like in CSNs with saturating synapses, using the approximated decoding Equation (3.12) eliminates the biologically implausible \( V \)-dependencies in the interaction terms. For an illustration of this coding see Fig. 3.1d.

3.3.2 Learning universal computations

Recurrent continuous rate networks are a powerful means for learning of various kinds of computations, like steering of movements and processing of sequences [JH04, SA09]. For this, an input and/or an output feedback signal needs to be able to “enslave” the network’s high-dimensional dynamics [Jae01, YJK12]. This means that at any point in time the network’s state is a deterministic function of the recent history of input and feedback signals. The function needs to be high dimensional, nonlinear, and possess fading memory. A standard model generating suitable dynamics are continuous rate networks of the form Equation (3.11). Due to the typically assumed random recurrent connectivity, each neuron acts as a randomly chosen, nonlinear function with fading memory. Linearly combining them like basis functions by a linear readout can approximate arbitrary, nonlinear functions with fading memory (time-scales are limited by the memory of the network), and in this sense universal computations on the input and the feedback. The feedback can prolong the fading memory and allow to generate self-contained dynamical systems and output sequences [JH04, MJS07, SA09, SA12]. The feedback can be incorporated into the network by directly training the recurrent synaptic weights [SA09, SA12].

Our understanding of the complex spiking dynamics of CSNs in terms of non-linear first order differential equations enables us to apply the above theory to spiking
neural networks: In the first step, we were able to conclude that our CSNs can generate enslaveable and thus computationally useful dynamics as they can be decoded to continuous dynamics that possess this property. In the second step, we have to ask which and how output signals should be learned to match a desired signal: In a biological setting, the appropriate signals are the sums of synaptic or dendritic input currents that spike trains generate, since these affect the somata of postsynaptic neurons as well as effectors such as muscles [EA03]. To perform, e.g., a desired continuous movement, they have to prescribe the appropriate muscle contraction strengths. For both CSNs with saturating synapses and with nonlinear dendrites, we choose the outputs to have the same form as the recurrent inputs that a soma of a neuron within the CSN receives. Accordingly, in our CSNs with saturating synapses, we interpret sums of the postsynaptic currents

\[ z_k(t) = \sum_{m=1}^{N} w^o_{km} \tanh(\gamma r_m(t)) =: \sum_{m=1}^{N} w^o_{km} \tilde{r}_m(t) \]  

(3.13)

as output signals, where the index \( k \) distinguishes \( K_{out} \) different outputs, and \( w^o_{km} \) are the learnable synaptic output weights. For networks with nonlinear dendrites the outputs are a linear combination of inputs preprocessed by nonlinear dendrites

\[ z_k(t) = \sum_{j=1}^{J} w^o_{kj} \tanh \left( \sum_{m=1}^{N} \Gamma_{jm} r_m(t) \right) =: \sum_{j=1}^{J} w^o_{kj} \tilde{r}_j(t), \]  

(3.14)

where the strengths \( w^o_{kj} \) of the dendro-somatic coupling are learned [LMM08]. The networks can now learn the output weights such that \( z_k(t) \) imitates a target signal \( F_k(t) \), using standard learning rules for linear readouts (see Fig. 3.2a for an illustration). We employ the recursive least squares method [Hay02].

To increase the computational and learning abilities, the output signals should be fed back to the network as an (additional) input (Fig. 3.2b)

\[ I^f_{e,\beta}(t) = \sum_{k=1}^{K_{sat}} w^f_{\beta k} z_k(t) = \sum_{k=1}^{K_{sat}} w^f_{\beta k} \sum_{\rho} w^o_{k\rho} \tilde{r}_\rho(t), \]  

(3.15)

where each neuron receives a linear combination of the output signals \( z_k(t) \) with static feedback connection strengths \( w^f_{\beta k} \). Here and in the following Greek letter indices such as \( \beta, \rho \) range over all saturating synapses (\( \beta, \rho = 1, ..., N; \tilde{r}_\beta(t) = \tanh(\gamma r_\beta(t)) \)) in CSNs with saturating synapses, or over all nonlinear dendrites \( \beta, \rho = 1, ..., J; \tilde{r}_\beta(t) = \tanh\left(\sum_{m=1}^{N} \Gamma_{\beta m} r_m(t)\right) \)) in CSNs with nonlinear dendrites.

It often seems biologically more plausible not to assume a strong feedback loop that enslaves the recurrent network, but rather to train recurrent weights. Our CSNs allow for this (Fig. 3.2c): We can transform the learning of output weights in networks with feedback into mathematically equivalent learning of recurrent
connection strengths, between synapses (CSNs with saturating synapses) or dendrites (CSNs with nonlinear dendrites) and the soma [LMM08] (we learn $A_{nm}$, see Methods for details of the implementation). We note that approximating different dynamical systems, e.g. ones equivalent to Equation (3.11) but with the coupling matrix inside the nonlinearity [Hir89], may also in CSNs with nonlinear dendrites allow to learn synaptic weights in similar manner. We call CSNs with learning of outputs in presence of feedback, or with learning of recurrent connections plastic continuous signal coding spiking neural networks (PCSNs).

To learn feedback and recurrent connections, we use the FORCE imitation learning rule, which has recently been suggested for networks of continuous rate neurons [SA09, SA12]: We use fast online learning based on the recursive least squares rule of the output weights in order to ensure that the output of the network is similar to the desired output at all times. Since during training the output is ensured to be close to the desired one, it can be used as feedback to the network at all times. The remaining deviations from the desired output are expected to be particularly suited as training noise as they reflect the system’s inherent noise. As mentioned before, the feedback loop may be incorporated in the recurrent network connectivity. During training, the reservoir connections are then learned in a similar manner as the readout.

In the following, we show that our approach allows spiking neural networks to perform a broad variety of tasks. In particular, we show learning of desired self-sustained dynamics at a degree of difficulty that has, to our knowledge, previously only been accessible with continuous rate networks.

### 3.3.3 Applications

#### Self-sustained pattern generation

Animals including humans can learn a great variety of movements, from periodic patterns like gait or swimming, to much more complex ones like producing speech, generating chaotic locomotion [Gri06, LPK+14] or playing the piano. Moreover when an animal learns to use an object (Fig. 3.3a), it has to learn the dynamical properties of the object as well as how its body behaves when interacting with it. Especially for complex, non-periodic dynamics, a dynamical system has to be learned with high precision.

How are spiking neural networks able to learn dynamical systems, store them and replay their activity? We find that PCSNs may solve the problem. They are able to learn periodic patterns of different degree of complexity as well as chaotic dynamical systems by imitation learning. Fig. 3.3 illustrates this for PCSNs with nonlinear synapses (Fig. 3.3d,e) and with nonlinear dendrites (Fig. 3.3b,e,f).

The figure displays the recall of three periodic movements after learning: a sine wave, a more complicated non-differentiable saw tooth pattern and a “camel’s hump” superposition of sine and cosine. Also for long simulation times, we find no deviation
3. Learning universal computations with spikes

**Figure 3.2**: Setups used to learn versatile nonlinear computations with spiking neural networks. (a) A static continuous signal coding spiking neural network (CSN, gray shaded) serves as a spiking computational reservoir with high signal-to-noise ratio. The results of computations on current and past external inputs $I_e$ can be extracted by simple neuron-like readouts. These linearly combine somatic inputs generated by saturating synapses or nonlinear dendrites, $\tilde{r}$ (red), to output signals $z$ (Equations (3.13, 3.14)). The output weights $w_o$ are learned such that $z$ approximates the desired continuous target signals. (b) Plastic continuous signal coding spiking neural networks (PCSNs) possess a loop that feeds the outputs $z$ back via static connections as an additional input $I_f$ (blue, Equation 3.15). Such networks have increased computational capabilities allowing them to, e.g., generate desired self-sustained activity. (c) The feedback loop can be incorporated into the recurrent network via plastic recurrent connections (red in gray shaded area).

from the displayed dynamics except for an inevitable phase shift (Fig. 3.12a in the supplement 3.6). It results from accumulation of small differences between the learned and desired periods. Apart from this, the error between the recalled and the desired signals is approximately constant over time (Fig. 3.12b in the supplement 3.6). This indicates that the network has learned a stable periodic orbit to generate the desired dynamics, the orbit is sufficiently stable to withstand the intrinsic noise of the system. Fig. 3.3 furthermore illustrates learning of a chaotic dynamical system. Here, the network learns to generate the time varying dynamics of all three components of the Lorenz system and produces the characteristic attractor pattern after learning (Fig. 3.3f). Due to the encoding of the dynamics in spike trains, the signal maintains a small deterministic error which emerges from the encoding of a continuous signal by discrete spikes (Fig. 3.3g). The individual training and recall trajectories quickly depart from each other after the end of learning since they are chaotic. However, also for long simulation times, we observe qualitatively the same dynamics, indicating that the correct dynamical system was learned (Fig. 3.12c in the supplement 3.6). Occasionally, errors occur, cf. the larger loop in Fig. 3.3f. This is to be expected due to the relatively short training period, during which only a part of the phase space covered by the attractor is visited. Importantly, we observe that after errors the dynamics return to the desired ones indicating that the general stability property of the attractor is captured by the learned system. To further test these observations, we considered a not explicitly trained long-term feature of the Lorenz-dynamics, namely the tent-map which relates the height $z_{n-1}$ of the
(n − 1)th local maximum in the z-coordinate, to the height $z_n$ of the subsequent local maximum. The spiking network indeed generates the map (Fig. 3.3h), with two outlier points corresponding to each error.

In networks with saturating synapses, the spike trains are characterized by possibly intermittent periods of rather high-frequency spiking. In networks with nonlinear dendrites, the spike trains can have low frequencies and they are highly irregular (Figs. 3.3c, 3.11 in the supplement 3.6). In agreement with experimental observations (e.g. [QRK+05]), the neurons can have preferred parts of the encoded signal in which they spike with increased rates.

The dynamics of the PCSNs and the generation of the desired signal are robust against dynamic and structural perturbations. They sustain noise inputs which would accumulate to several ten percent of the level of the threshold within the membrane time constant, for a neuron without further input (Fig. 3.7 in the supplement 3.6). For larger deviations of $W_{njm}$ from their optimal values, PCSNs with nonlinear dendrites can keep their learning capabilities, if $\mu$ is tuned to a specific range. Outside this range, the capabilities break down at small deviations (Fig. 3.8 in the supplement 3.6). However, a slightly modified version of the models, where the reset is always to $-\theta$ (even if there was fast excitation that drove the neuron to spike by a suprathreshold input), has a high degree of robustness against such structural perturbations. We also checked that the fast connections are important, albeit substantial weakening can be tolerated (Fig. 3.9 in the supplement 3.6).

The deterministic spike code of our PCSNs encodes the output signal much more precisely than neurons generating a simple Poisson code, which facilitates learning. We have quantified this using a comparison between PCSNs with saturating synapses and networks of Poisson neurons of equal size, both learning the saw tooth pattern in the same manner. Since both codes become more precise with increasing spike rate of individual neurons, we compared the testing error between networks with equal spike rates. Due to their higher signal-to-noise ratio, firing rates required by the PCSNs to achieve the same pattern generation quality are more than one order of magnitude lower (Fig. 3.6 in the supplement 3.6).

### Delayed reaction/time interval estimation

For many tasks, e.g. computations focusing on recent external input and generation of self-sustained patterns, it is essential that the memory of the involved recurrent networks is fading: If past states cannot be forgotten, they lead to different states in response to similar recent inputs. A readout that learns to extract computations on recent input will then quickly reach its capacity limit. In neural networks, fading memory originates on the one hand from the dynamics of single neurons, e.g. due to their finite synaptic and membrane time constants; on the other hand it is a consequence of the neurons’ connection to a network [TWG04, WLS04, Gol09]. In standard spiking neural network models, the overall fading memory is short, of the order of hundreds of milliseconds [IS04, JM05, MG05, WML13]. It is a matter
3. Learning universal computations with spikes

**Figure 3.3: Learning dynamics with spiking neural networks.** (a): Schematic hunting scene, illustrating the need for complicated dynamical systems learning and control. The hominid has to predict the motion of its prey, and to predict and control the movements of its body and the projectile. (b-h): Learning of self-sustained dynamical patterns by spiking neural networks. (b): A sine wave generated by summed, synaptically and dendritically filtered output spike trains of a PCSN with nonlinear dendrites. (c): A sample of the network’s spike trains generating the sine in (b). (d): A saw tooth pattern generated by a PCSN with saturating synapses. (e): A more complicated smooth pattern generated by both architectures (blue: nonlinear dendrites, red: saturating synapses). (f-h): Learning of chaotic dynamics (Lorenz system), with a PCSN with nonlinear dendrites. (f): The spiking network imitates an example trajectory of the Lorenz system during training (blue); it continues generating the dynamics during testing (red). (g): Detailed view of (f) highlighting how the example trajectory (yellow) is imitated during training and continued during testing. (h): The spiking network approximates not explicitly trained quantitative dynamical features, like the tent map between subsequent maxima of the z-coordinate. The ideal tent map (yellow) is closely approximated by the tent map generated by the PCSN (red). The spiking network sporadically generates errors, cf. the larger loop in (f) and the outlier points in (h). Panel (h) shows a ten times longer time series than (f), with three errors.
of current debate how this can be extended by suitable single neuron properties and topology [MNM02, BMD13, HVG14, Ost14]. Many biological computations, e.g. the simple understanding of a sentence, require longer memory, on the order of seconds.

We find that CSNs without learning of recurrent connectivity or feedback access such time scales. We illustrate this by means of a delayed reaction/time estimation task: In the beginning of a trial, the network receives a short input pulse. By imitation learning, the network output learns to generate a desired delayed reaction. For this, it needs to specifically amplify the input’s dynamical trace in the recurrent spiking activity, at a certain time interval. The desired response is a Gaussian curve, representative for any type of delayed reaction. The reaction can be generated several seconds after the input (Fig. 3.4a-c).

The quality of the reaction pattern depends on the connection strengths within the network, specified by the spectral radius \( g \) of the coupling matrix divided by the leak of a single corresponding continuous unit \( \lambda_x \). Memory is kept best in an intermediate regime (Fig. 3.4b), where the CSN stays active over long periods of time without overwriting information. This has also been observed for continuous rate networks [SCS88]. For too weak connections (Fig. 3.4a), the CSN returns to the inactive state after short time, rendering it impossible to retrieve input information later. If the connections are too strong, (Fig. 3.4c), the CSN generates self-sustained, either irregular asynchronous or oscillating activity, partly overwriting information and hindering its retrieval. We observe that already the memory in disconnected CSNs with synaptic saturation can last for times beyond hundreds of milliseconds (cf. Fig. 3.10 in the supplement 3.6). This is a consequence of the recovery current: If a neuron has spiked several times in succession, the accumulated recovery current leads to further spiking (and further recovery current), and thus dampens the decay of a strong activation of the neuron [SLRT00].

Experiments show that during time estimation tasks, neurons are particularly active at two times: When the stimulus is received and when the estimated time has passed [DH06, MMN03]. Often the neuron populations that show activity at these points are disjoint. Our model reproduces this behavior for networks with good memory performance. In particular, at the time of the initial input the recurrently connected neurons become highly active (gray traces in Fig. 3.4b, upper sub-panel) while at the estimated reaction time, readout neurons would show increased activity (red trace).
3. Learning universal computations with spikes
Figure 3.4: Learning of longer-term memory dependent computations with spiking neural networks. (a-c): Delayed reaction and time interval estimation: The synaptic output of a CSN learns to generate a generic reaction several seconds after a short input. Upper panels show typical examples of input, desired and actual reactions (green, yellow and red traces). In the three panels, the desired reaction delay is the same (9 sec), the networks (CSNs with saturating synapses) have different levels of recurrent connection strengths ((a), (b), (c): low, intermediate, high level). The generation of the reaction is best for the network with intermediate level of connection strength. The CSNs with lower or higher levels have not maintained sufficient memory due to their extinguished or noisy and likely chaotic dynamics (gray background lines: spike rates of individual neurons). The median errors of responses measured for different delays in ensembles of networks (levels of connection strength as in the upper panels), are given in the lower panels. The shaded regions represent the area between the first and third quartile of the response errors. Dashed lines highlight delay and error size of the examples in the upper panels. (d): Persistent retaining of instructions and switching between computations: The network receives (i) two random continuous operand inputs (upper sub-panel, yellow and purple traces), and (ii) two pulsed instruction inputs (middle sub-panel, blue and green; memory of last instruction pulse: red). The network has learned to perform different computations on the operand inputs, depending on the last instruction (lower sub-panel): if it was +1 (triggered by instruction channel 1), the network performs a nonlinear computation, it outputs the absolute value of the difference of the operands (red trace (network output) agrees with blue); if it was -1 (triggered by channel 2), the values of the operands are added (red trace agrees with green trace).
3. Learning universal computations with spikes

**Persistent memory and context dependent switching**

Tasks often also require to store memories persistently, e.g. to remember instructions [SP03]. Such memories may be maintained in learned attractor states (e.g. [BT14, Hop82, LKD14, ZAG15]). In the framework of our computing scheme, this requires the presence of output feedback [MJS07]. Here, we illustrate the ability of PCSNs to learn and maintain persistent memories as attractor states as well as the ability to change behavior according to them. For this, we use a task that requires memorizing computational instructions (Fig. 3.4d) [MJS07]. The network has two types of inputs: After pulses in the instruction channels, it needs to switch persistently between different rules for computation on the current values of operand channels. To store persistent memory, the recurrent connections are trained such that an appropriate output can indicate the instruction channel that has sent the last pulse: The network learns to largely ignore the signal when a pulse arrives from the already remembered instruction channel, and to switch states otherwise. Due to the high signal-to-noise ratio of our deterministic spike code, the PCSNs are able to keep a very accurate representation of the currently valid instruction in their recurrent dynamics. Fig. 3.4d, middle sub-panel, shows this by displaying the output of the linear readout trained to extract this instruction from the network dynamics. A similarly high precision can be observed for the output of the computational task, cf. Fig. 3.4d, lower sub-panel.

**Building of world models, and control**

In order to control its environment, an animal has to learn the laws that govern the environment’s dynamics, and to develop a control strategy. Since environments are partly unpredictable and strategies are subject to evolutionary pressure, we expect that they may be described by stochastic optimal control theory. A particularly promising candidate framework is path integral control, since it computes the optimal control by simulating possible future scenarios under different random exploratory controls, and the optimal control is a simple weighted average of them [Kap05]. For this, an animal needs an internal model of the system or tool it wants to act on. It can then mentally simulate different ways to deal with the system and compute an optimal one. Recent experiments indicate that animals indeed conduct thought experiments exploring and evaluating possible future actions and movement trajectories before performing one [PF13, VDMR10].

Here we show that by imitation learning, spiking neural networks, more precisely PCSNs with a feedback loop, can acquire an internal model of a dynamical system and that this can be used to compute optimal controls and actions. As a specific, representative task, we choose to learn and control a stochastic pendulum (Fig. 3.5a,b). The pendulum’s dynamics are given by

\[
\ddot{\phi}(t) + c\omega_0\dot{\phi}(t) + \omega_0^2 \sin(\phi(t)) = \xi(t) + u(t),
\]  

(3.16)
with the angular displacement $\phi$ relative to the direction of gravitational acceleration, the undamped angular frequency for small amplitudes $\omega_0$, the damping ratio $c$, a random (white noise) angular force $\xi(t)$ and the deterministic control angular force $u(t)$, both applied to the pivot axis. The PCSN needs to learn the pendulum’s dynamics under largely arbitrary external control forces; this goes beyond the tasks of the previous sections. It is achieved during an initial learning phase characterized by motor babbling as observed in infants [vH82] and similarly in bird song learning [FFS07]: During this phase, there is no deterministic control, $u = 0$, and the pendulum is driven by a random exploratory force $\xi$ only. Also the PCSN receives $\xi$ as input and learns to imitate the resulting pendulum’s dynamics with its output.

During the subsequent control phase starting at $t = 0$, the aim is to swing the pendulum up and hold it in the inverted position (Fig. 3.5c). For this, the PCSN simulates at time $t$ a set of $M$ future trajectories of the pendulum, for different random exploratory forces $\xi_i$ (“mental exploration” with $u = 0$, cf. Fig. 3.5a,b), starting with the current state of the pendulum. In a biological system, the initialization may be achieved through sensory input taking advantage of the fact that an appropriately initialized output enslaves the network through the feedback. Experiments indicate that explored trajectories are evaluated, by brain regions separate from the ones storing the world model [LGL+08,vR09,LGL+09]. We thus assign to the simulated trajectories a reward $R_i$ measuring the agreement of the predicted states with the desired ones. The optimal control $u(t + s)$ (cf. Equation (3.16)) for a subsequent, not too large time interval $s \in [0, \delta]$ is then approximately given by a temporal average over the initial phase of the assumed random forces, weighted by the exponentiated total expected reward,

$$u(t + s) = \sum_{i=1}^{M} \frac{e^{\lambda_i R_i(t)}}{\sum_{j=1}^{M} e^{\lambda_j R_j(t)}} \bar{\xi}_i(t),$$  

(3.17)

where $\bar{\xi}_i(t) = \frac{1}{\delta} \int_{t}^{t+\delta} \xi_i(\tilde{t}) d\tilde{t}$ and $\lambda_c$ is a weighting factor. We have chosen $R_i(t) = \int_{t}^{t+T_r} y_i(\tilde{t}) d\tilde{t}$, i.e. the expected reward increases linearly with the heights $y_i(\tilde{t}) = -\cos(\phi_i(\tilde{t}))$ predicted for the pendulum for input trajectory $\xi_i$; it becomes maximal for a trajectory at the inversion point. $T_r$ is the duration of a simulated trajectory. The optimal control is applied to the pendulum until $t + \Delta$, with $\Delta < \delta$. Then, at $t + \Delta$, the PCSN simulates a new set of trajectories starting with the pendulum’s updated state and a new optimal control is computed. This is valid and applied to the pendulum between $t + \Delta$ and $t + 2\Delta$, and so on. We find that controlling the pendulum by this principle leads to the desired upswing and stabilization in the inversion point, even though we assume that the perturbing noise force $\xi$ (Equation (3.16)) acting on the pendulum in addition to the deterministic control $u$, remains as strong as it was during the exploration/learning phase (cf. Fig. 3.5a,b).

We find that for controlling the pendulum, the learned internal model of the
3. Learning universal computations with spikes

Figure 3.5: Model building and mental exploration to compute optimal control. (a): Learning of an internal world model with spiking neural networks. During model building, random exploratory control drives the dynamical system (here: a swinging pendulum). The spiking neural network is provided with the same control as input and learns to mimic the behavior of the pendulum as its output. (b): After learning, the spiking network can simulate the system’s response to control signals. The panel displays the height of the real pendulum in the past (solid black line) and future heights under different exploratory controls (dashed lines). For the same controls, the spiking neural network predicts very similar future positions (colored lines) as the imitated system. It can therefore be used for mental exploration and computation of optimal control to reach an aim, here: to invert the pendulum. (c): During mental exploration, the network simulates in regular time intervals a set of possible future trajectories for different controls, starting from the actual state of the pendulum. From this, the optimal control until the next exploration can be computed and applied to the pendulum. The control reaches its aim: The pendulum is swung up and held in inverted position, despite a high level of noise added during testing (uncontrolled dynamics as in panel (a)).
system has to be very accurate. This implies that particular realizations of the PCSN can be unsuited to learn the model (we observed this for about half of the realizations), a phenomenon that has also been reported for small continuous rate networks before. However, we checked that continuous rate networks as encoded by our spiking ones reliably learn the task. Since the encoding quality increases with the number of spiking neurons, we expect that sufficiently large PCSNs reliably learn the task as well.

3.4 Discussion

The characteristic means of communication between neurons in the nervous system are spikes. It is widely accepted that sequences of spikes form the basis of neural computations in higher animals. How computations are performed and learned is, however, largely unclear. Here we have derived continuous signal coding spiking neural networks (CSNs), a class of mesoscopic spiking neural networks that are a suitable substrate for computation. Together with plasticity rules for their output or recurrent connections, they are able to learn general, complicated computations by imitation learning (plastic CSNs, PCSNs). Learning can be highly reliable and accurate already for comparably small networks of hundreds of neurons. The underlying principle is that the networks reflect the input in a complicated nonlinear way, generate nonlinear transformations of it and use fading memory such that the inputs and their pasts interfere with each other. This requires an overall nonlinear relaxation dynamics suitable for computations [JH04]. Such dynamics are different from standard spiking neural network dynamics, which are characterized by a high level of noise and short intrinsic memory [JM05, MG05, JMT09, WML13].

To find spiking networks that generate appropriate dynamics, we use a linear decoding scheme for continuous signals encoded in the network dynamics as combinations of membrane potentials and synaptic currents. A specific coding scheme like this was introduced in refs. [BD11, BMD13] to derive spiking networks encoding linear dynamics in an optimal way. We introduce spiking networks where the encoded signals have dynamics desirable for computation, i.e. a nonlinear, high-dimensional, low-noise, relaxational character as well as significant fading memory. We conclude that, since we use simple linear decoding, already the dynamics of the spiking networks must possess these properties.

Using this approach, we study two types of CSNs: Networks with saturating synapses and networks with nonlinear dendrites. The CSNs with saturating synapses use a direct signal encoding; each neuron codes for one continuous variable. It requires spiking dynamics characterized by possibly intermittent phases of high rate spiking, or bursting, with inter-spike-intervals smaller than the synaptic time constants, which leads to a temporal averaging over spikes. Dynamics that appear externally similar to such dynamics were recently highlighted as a ‘second type of balanced state’ in networks of pulse-coupled, intrinsically oscillating model.
neurons [Ost14]. Very recently [HH15, KS15] showed that networks whose spiking dynamics are temporally averaged due to slow synapses possess a phase transition from a fixed point to chaotic dynamics in the firing rates, like the corresponding rate models that they directly encode. In the analytical computations the spike coding was not specified [HH15] or assumed to be Poissonian [KS15]. Numerical simulations of leaky integrate-and-fire neurons in the chaotic rate regime can generate intermittent phases of rather regular high-rate spiking [HH15]. The networks might provide a suitable substrate for learning computations as well. However, since the chaotic rate dynamics have correlations on the time scale of the slow synapses its applicability is limited to learning tasks where only a short fading memory of the reservoir is needed. For example delayed reaction tasks as illustrated in Fig. 3.4a-c would not be possible. Interestingly, in our scheme a standard leaky integrate-and-fire neuron with saturating synapses appears as a special case with recovery current of amplitude zero. According to our analysis it can act as a leaky integrator with a leak of the same time constant as the synapses, $\lambda_x = \lambda_s$. In contrast, in presence of a recovery current, our networks with saturating synapses can encode slower dynamics on the order of seconds. After training the network, the time scales can be further extended.

In the CSNs with nonlinear dendrites the entire neural population codes for a usually smaller number of continuous variables, avoiding high firing rates in sufficiently large networks. The networks generate irregular, low frequency spiking and simultaneously a noise-reduced encoding of nonlinear dynamics, the temporal averaging over spikes in the direct coding case is partially replaced by a spatial averaging over spike trains from many neurons. The population coding scheme and our derivations of CSNs with nonlinear dendrites generalize the predictive coding proposed in ref. [BMD13] to nonlinear dynamics. The roles of our slow and fast connections are similar to those used there: In particular, redundancies in the spiking are eliminated by fast recurrent connections without synaptic filtering. We expect that these couplings can be replaced by fast connections that have small finite synaptic time constants, as shown for the networks of ref. [BMD13] in ref. [SFDSB15]. In contrast to previous work, in the CSNs with nonlinear dendrites we have linear and nonlinear slow couplings. The former contribute to coding precision and implement linear parts of the encoded dynamics, the latter implement the nonlinearities in the encoded dynamics. Further, in contrast to previous work, the spike coding networks provide only the substrate for learning of general dynamical systems by adapting their recurrent connections. Importantly, this implies (i) that the neurons do not have to adapt their nonlinearities to each nonlinear dynamical system that is to be learned (which would not seem biologically plausible) and (ii) that the CSNs do not have to provide a faithful approximation of the nonlinear dynamics Equations (3.6),(3.11), since a rough dynamical character (i.e. slow dynamics and the echo state property) is sufficient for serving as substrates. We note that refs. [Eli05, ESC’12] suggested to use the differential equations that
characterize dynamical systems to engineer spiking neural networks that encode the
dynamics. The approach suggests an alternative derivation of spiking networks that
may be suitable as substrate for learning computations. Their rate coding scheme,
however, allows for redundancy and thus higher noise levels, and it generates high
frequency spiking. In a future publication, B. DePasquale, M. Churchland, and
L.F. Abbott will present an approach to train rate coding spiking neural networks,
with continuous rate networks providing the target signals [DCA16]. We will discuss
the relation between our and this approach in a joint review [ADM16].

A characteristic feature of our neuron models is that they take into account
nonlinearities in the synapses or in the dendrites. On the one hand this is biologically
plausible [BFR04, LH05, Mem10, CHG13], on the other hand it is important for
generating nonlinear computations. Our nonlinearities are such that the decoded
continuous dynamics match those for typical networks of continuous rate neurons
and provide a simple model for dendritic and synaptic saturation. However, the
precise form of the neuron model and its nonlinearity is not important for our
approaches: As long as the encoded dynamical system is suitable as a computational
reservoir, the spiking system is a CSN and our learning schemes will work. As an
example, a dendritic tree with multiple interacting compartments may be directly
implemented in both the networks with saturating synapses and in the networks
with nonlinear dendrites. A future task is to explore the computational capabilities
of CSNs incorporating different and biologically more detailed features that lead
to nonlinearities, e.g. neural refractory periods, dendritic trees with calcium and
NMDA voltage dependent channels and/or standard types of short term synaptic
plasticity.

Inspired by animals’ needs to generate and predict continuous dynamics such
as their own body and external world movements, we let our networks learn to
approximate desired continuous dynamics. Since effector organs such as muscles
and post-synaptic neurons react to weighted, possibly dendritically processed sums
of post-synaptic currents, we interpret these sums as the relevant, continuous signal-
approximating outputs of the network [EA03]. Importantly, this is not the same
as Poissonian rate coding of a continuous signal: As a simple example, consider a
single spiking neuron. In our scheme it will spike with constant inter-spike-intervals
to encode a constant output. In Poissonian rate coding, the inter-spike-intervals will
be random, exponentially distributed and many more spikes need to be sampled to
decode the constant output (cf. Fig. 3.6 in the supplement 3.6).

The outputs and recurrent connections of CSNs can be learned by standard
learning rules [Hay02, SA09]. The weight changes depend on the product of the
error and the synaptic or dendritic currents and may be interpreted as delta-
rules with synapse- and time-dependent learning rates. PCSNs, with learning of
recurrent weights or output feedback, show how spiking neural networks may
learn internal models of complicated, self-sustained environmental dynamics. In
our applications, we demonstrate that they can learn to generate and predict the
dynamics in different depths, ranging from the learning of single stable patterns over the learning of chaotic dynamics to the learning of dynamics incorporating their reactions to external influences.

The spiking networks we use have medium size, like networks with continuous neurons used in the literature [JH04, SA09]. CSNs with saturating synapses have, by construction, the same size as their non-spiking counterparts. In CSNs with nonlinear dendrites the spike load necessary to encode the continuous signals is distributed over the entire network. This leads to a trade-off between lower spiking frequency per neuron and larger network size (cf. Fig. 3.11 in the supplement 3.6): The faster the neurons can spike the smaller the network may be to solve a given task.

Previous work using spiking neurons as a reservoir to generate a high dimensional, nonlinear projection of a signal for computation, concentrated on networks without output feedback or equivalent task-specific learning of recurrent connectivity [MNM02, LM07, HVG14]. Such networks are commonly called “liquid state machines” [Maa10]. By construction, they are unable to solve tasks like the generation of self-sustained activity and persistent memorizing of instructions; these require an effective output feedback, since the current output determines the desired future one: To compute the latter, the former must be made available to the network as an input. The implementation of spiking reservoir computers with feedback was hindered by the high level of noise in the relevant signals: The computations depend on the spike rate, the spike trains provide a too noisy approximation of this average signal and the noise is amplified in the feedback loop. While analytically considering feedback in networks of continuous rate neurons, ref. [MJS07] showed examples of input-output tasks solved by spiking networks with a feedback circuit, the output signals are affected by a high level of noise. This concerns even output signals just keeping a constant value. We implemented similar tasks (Fig. 3.4d), and find that our networks solve them very accurately due to their more efficient coding and the resulting comparably high signal-to-noise ratio. In contrast to previous work, our derivations systematically delineate spiking networks which are suitable for the computational principle with feedback or recurrent learning; the networks can accurately learn universal, complicated memory dependent computations as well as dynamical systems approximation, in particular the generation of self-sustained dynamics.

In the control task, we show how a spiking neural network can learn an internal model of a dynamical system, which subsequently allows to control the system. We use a path integral approach, which has already previously been suggested as a theory for motor control in biological systems [Fri11, Tod09b]. We apply it to learned world models, and to neural networks. Path integral control assumes that noise and control act in a similar way on the system [Kap05]. This assumption is comparably weak and the path integral control method has been successfully applied in many robotics applications [TBS10a, PKC+11, BTSS10], where it was
found to be superior to reinforcement learning and adaptive control methods. Continuous rate networks using recurrence, readouts, and feedback or equivalent recurrent learning, are versatile, powerful devices for nonlinear computations. This has inspired their use in manifold applications in science and engineering, such as control, forecasting and pattern recognition [LJS12]. Our study has demonstrated that it is possible to obtain similar performance using spiking neural networks. Therewith, our study makes spiking neural networks available for similarly diverse, complex computations and supports the feasibility of the considered computational principle as a principle for information processing in the brain.

3.5 Methods

3.5.1 Network simulation

We use a time grid based simulation scheme (step size \(dt\)). If not mentioned otherwise, between time points, we compute the membrane potentials using a Runge-Kutta integration scheme for dynamics without noise and an Euler-Maruyama integration scheme for dynamics with noise. Since CSNs with nonlinear dendrites have fast connections without conduction delays and synaptic filtering, we process spikings at a time point as follows: We test whether the neuron with the highest membrane potential is above threshold. If the outcome is positive, the neuron is reset and the impact of the spike on postsynaptic neurons is evaluated. Thereafter, we compute the neuron with the highest, possibly updated, membrane potential and repeat the procedure. If all neurons have subthreshold membrane potential, we proceed to the next time point. The described consecutive updating of neurons in a single time step increases in networks with nonlinear dendrites the robustness of the simulations against larger time steps, as the neurons maintain an order of spiking and responding like in a simulation with smaller time steps and a small but finite conduction delay and/or slight filtering of fast inputs. As an example, the scheme avoids that neurons that code for similar features and thus possess fast mutual inhibition, spike together within one step and generate an overshoot in the readout, as it would be the case in a parallel membrane potential updating scheme. The different tasks use either networks with saturating synapses or networks with nonlinear dendrites. In both cases, \(A\) is a sparse matrix with a fraction \(p\) of non-zero values. These are drawn independently from a Gaussian distribution with zero mean and variance \(\frac{\sigma^2}{pN}\) (CSNs with saturating synapses) or \(\frac{\sigma^2}{pJ}\) (CSNs with nonlinear dendrites), which sets the spectral radius of \(A\) approximately to \(g\). For networks with nonlinear dendrites, the elements of \(\Gamma\) are drawn from a standard normal distribution. To keep the approach simple, we allow for positive and negative dendro-somatic couplings. In order to achieve a uniform distribution of spiking over the neurons in the network, we normalize the columns of \(\Gamma\) to have the same norm, which we control with the parameter \(\gamma_s\). This implies that the thresholds are
3. Learning universal computations with spikes

identical.

3.5.2 Training phase

The networks are trained for a period of length $T$, such that the readouts $z_k$ imitate target signals $F_k(t)$, i.e. such that the time average of the square of the errors $e_k(t) = z_k(t) - F_k(t)$ is minimized. At $T$, training stops and the weights are not updated anymore in the subsequent testing. If present, the external input to the neurons is a weighted sum of $K_{in}$ continuous input signals $f_k(t)$, $I_{e,\beta}(t) = \sum_{k=1}^{K_{in}} w_{\beta k} f_k(t)$, where the index $\beta$ runs from 1 to $N$ (CSNs with saturating synapses) or from 1 to $J$ (CSNs with nonlinear dendrites). The weights $w_{\beta k}$ are fixed and drawn from a uniform distribution in the range $[-\tilde{w}^i, \tilde{w}^i]$. If present, the feedback weights $w_{\beta k}^{f}$ (cf. Equation (3.15)) are likewise chosen randomly from a uniform distribution in the range $[-e^{w}_{f}, e^{w}_{f}]$ with a global feedback parameter $e^{w}_{f}$.

For the delayed reaction/time estimation task (Figs. 3.4a-c, 3.10 in the supplement 3.6), we applied the RLS (recursive least squares) algorithm [Hay02] to learn the linear outputs. For the pattern generation, instruction switching and control tasks, we applied the FORCE (first-order reduced and controlled error) algorithm [SA09] (Figs. 3.3, 3.4d, 3.5, 3.6-3.9, 3.11 and 3.12 in the supplement 3.6) to learn the recurrent connections and linear outputs.

3.5.3 Learning rules

The output weights $w_{km}^{o}$ are trained using the standard recursive least squares method [Hay02]. They are initialized with 0, we use weight update intervals of $\Delta t$. The weight update uses the current training error $e_k(t) = z_k(t) - F_k(t)$, where $z_k(t)$ is the output that should imitate the target signal $F_k(t)$, it further uses an estimate $P_{\beta\gamma}(t)$ of the inverse correlation matrix of the unweighted neural synaptic or dendritic inputs $\tilde{r}_{\beta}(t)$, as well as these inputs,

$$w_{k\beta}^{o}(t) = w_{k\beta}^{o}(t - \Delta t) - e_k(t) \sum_{\rho} P_{\beta\rho}(t) \tilde{r}_{\rho}(t).$$

(3.18)

The indices $\beta, \rho$ range over all saturating synapses ($\beta, \rho = 1, ..., N$; $\tilde{r}_{\beta}(t) = \tanh(\gamma r_{\beta}(t))$) or all non-linear dendrites ($\beta, \rho = 1, ..., J$; $\tilde{r}_{\beta}(t) = \tanh(\sum_{m=1}^{N} \Gamma_{\beta m} r_{m}(t))$) of the output neuron. The square matrix $P$ is a running filter estimate of the inverse correlation matrix of the activity of the saturated synapses (CSNs with saturating synapses) or non-linear dendrites (CSNs with nonlinear dendrites). The matrix is updated via

$$P_{\beta\gamma}(t) = P_{\beta\gamma}(t - \Delta t) - \frac{\sum_{\rho} \sum_{\sigma} P_{\beta\rho}(t - \Delta t) \tilde{r}_{\rho}(t) \tilde{r}_{\sigma}(t) P_{\sigma\gamma}(t - \Delta t)}{1 + \sum_{\rho} \sum_{\sigma} \tilde{r}_{\rho}(t) P_{\rho\sigma}(t - \Delta t) \tilde{r}_{\sigma}(t)},$$

(3.19)
where the indices $\beta, \gamma, \rho, \sigma$ run from 1 to $N$ (CSNs with saturating synapses) or from 1 to $J$ (CSNs with nonlinear dendrites). $P$ is initialized as $P(0) = \alpha^{-1}1$ with $\alpha^{-1}$ acting as a learning rate.

For the update of output weights in presence of feedback and of recurrent weights we adopt the FORCE algorithm \cite{SA09}. In presence of feedback, this means that recursive least squares learning of output is fast against the temporal evolution of the network, and already during training the output is fed back into the network. Thus, each neuron gets a feedback input

$$I_{e,\beta}^f(t) = \sum_{k=1}^{K_{out}} w_{\beta k}^f z_k(t) = \sum_{k=1}^{K_{out}} w_{\beta k}^f \sum_{\rho} w_{\rho k}^o \tilde{r}_\rho(t).$$  \hspace{1cm} (3.20)

The feedback weights $w_{\beta k}^f$ are static, the output weights are learned according to Equation (3.18).

Since the outputs are linear combinations of synaptic or dendritic currents, which also the neurons within the network linearly combine, the feedback loop can be implemented by modifying the recurrent connectivity, by adding a term $\sum_{k=1}^{K_{out}} w_{\rho k}^f w_{\rho k}^o$ to the matrix $A_{\rho \beta}$. Learning then affects the output weights as well as the recurrent connections, separate feedback connections are not present. This learning and learning of output weights with a feedback loop are just two different interpretations of the same learning rule. For networks with saturating synapses the update is

$$A_{nm}(t) = A_{nm}(t-\Delta t) - \sum_{k=1}^{K_{out}} w_{nk}^f e_k(t) \sum_{l=1}^{N} P_{ml}(t) \tilde{r}_l(t),$$  \hspace{1cm} (3.21)

where the $w_{nk}^f$ are now acting as learning rates. For networks with nonlinear dendrites, the update is

$$D_{nj}(t) = D_{nj}(t-\Delta t) - \sum_{i=1}^{J} \Gamma_{in} \sum_{k=1}^{K_{out}} w_{ik}^f e_k(t) \sum_{h=1}^{J} P_{jh}(t) \tilde{r}_h(t).$$  \hspace{1cm} (3.22)

### 3.5.4 Control task

The task is achieved in two phases, the learning and the control phase.

1. Learning: The PCSN learns a world model of the noisy pendulum, i.e. it learns the dynamical system and how it reacts to input. The pendulum follows the differential Equation (3.16) with $c\omega_0 = 0.1s^{-1}$ and $\omega_0^2 = 10s^{-2}$, $\xi(t)$ is a white noise force with $\langle \xi(t)\xi(t') \rangle = s^{-2}\delta(t-t')$, $x(t) = \sin(\phi(t))$ and $y(t) = -\cos(\phi(t))$ are Cartesian coordinates of the point mass. The neural network has one input and three outputs which are fed back into the network; it learns to output the $x$- and the $y$-coordinate, as well as the angular velocity of the pendulum when it receives
3. Learning universal computations with spikes

as input the strength of the angular force (noise plus control) \( \xi(t) + u(t) \) applied to the pivot axis of the pendulum. The learning is here interpreted as learning in a network with feedback, cf. Equation (3.20).

We created a training trajectory of length \( T_t = 1000 \) s by simulating the pendulum with the given parameters and by driving it with white noise \( \xi(t) \) as an exploratory control (\( u(t) = 0 \)). Through its input, the PCSN receives the same white noise realization \( \xi(t) \). During training the PCSN learns to imitate the reaction of the pendulum to this control, more precisely its outputs learn to approximate the trajectories of \( x, y \) and \( \omega \). As feedback to the reservoir during training we choose a convex combination of the reservoir output and the target (feedback = 0.9 · output + 0.1 · target). We find that such a combination improves performance: If the output at the beginning of the training is very erroneous, those errors are accumulated through the feedback-loop, which prevents the algorithm from working. On the other hand, if one feeds back only the target signal, the algorithm does not learn how to correct for feedback transmitted readout errors. In our task, the convex combination alleviates both problems.

2. Control: In the second phase, the learned world model of the pendulum is used to compute stochastic optimal control that swings the pendulum up and keeps it in the inverted position. The PCSN does not learn its weights in this phase anymore. It receives the different realizations of exploratory (white noise) control and predicts the resulting motion (“mental exploration”). From this, the optimal control may be computed using the path integral framework [Kap05]. In this framework a stochastic dynamical system (which is possibly multivariate)

\[
\dot{x}(t) = f(x(t)) + u(x(t), t) + \xi(t)
\]  

(3.23)

with arbitrary nonlinearity \( f(x(t)) \) and white noise \( \xi(t) \), is controlled by the feedback controller \( u(x(t), t) \) to optimize an integral \( C(t) \) over a state cost \( U(x(\tilde{t})) \) and a moving horizon quadratic control cost, \( C(t) = \int_t^{t+T} U(x(\tilde{t})) + u(\tilde{t})^2 d\tilde{t} \). The reward is related to the cost by \( R = -C \). Path integral control theory shows that the control at time \( t \) can be computed by generating samples from the dynamical system under the uncontrolled dynamics

\[
\dot{x}(t) = f(x(t)) + \xi(t).
\]  

(3.24)

The control is then given by the success weighted average of the noise realizations \( \xi_i \)

\[
u(t) = \lim_{\delta \to 0} \lim_{M \to \infty} \sum_{i=1}^{M} \frac{e^{-\lambda_C C_i(t)}}{\sum_{j=1}^{M} e^{-\lambda_C C_j(t)}} \frac{1}{\delta} \int_t^{t+\delta} \xi_i(\tilde{t}) d\tilde{t},
\]  

(3.25)

where \( C_i(t) = \int_t^{t+T} U(x_i(\tilde{t})) d\tilde{t} \) is the cost observed in the \( i \)th realization of the uncontrolled dynamics, which is driven by noise realization \( \xi_i \) and \( u = 0 \). Equa-
3.5. Methods

<table>
<thead>
<tr>
<th>Sat. syn.</th>
<th>N</th>
<th>α</th>
<th>dt</th>
<th>(T_t)</th>
<th>(\lambda_s^{-1})</th>
<th>(\lambda_r^{-1})</th>
<th>(V_r)</th>
<th>(\theta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 3.3d</td>
<td>50</td>
<td>0.1</td>
<td>0.1ms</td>
<td>100s</td>
<td>100ms</td>
<td>100ms</td>
<td>0.9(\theta)</td>
<td>0.03</td>
</tr>
<tr>
<td>Fig. 3.3e</td>
<td>50</td>
<td>0.1</td>
<td>1ms</td>
<td>100s</td>
<td>100ms</td>
<td>100ms</td>
<td>0.9(\theta)</td>
<td>0.03</td>
</tr>
<tr>
<td>Fig. 3.4a-c</td>
<td>200</td>
<td>0.1</td>
<td>1ms</td>
<td>800s</td>
<td>100ms</td>
<td>50ms</td>
<td>0.54(\theta)</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**Table 3.2:** Parameters used in the different figures for simulations of networks with saturating synapses.

<table>
<thead>
<tr>
<th>Nonlin. dendr.</th>
<th>N</th>
<th>J</th>
<th>α</th>
<th>(\gamma_s)</th>
<th>dt</th>
<th>(T_t)</th>
<th>(\mu)</th>
<th>(\lambda_s^{-1})</th>
<th>(\lambda_r^{-1})</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 3.3b,c</td>
<td>500</td>
<td>50</td>
<td>0.1</td>
<td>0.03</td>
<td>1ms</td>
<td>100s</td>
<td>0</td>
<td>100ms</td>
<td>100ms</td>
<td>(\lambda_s^{-1}s)</td>
</tr>
<tr>
<td>Fig. 3.3e</td>
<td>500</td>
<td>50</td>
<td>0.1</td>
<td>0.03</td>
<td>1ms</td>
<td>100s</td>
<td>0</td>
<td>100ms</td>
<td>100ms</td>
<td>(\lambda_s^{-1}s)</td>
</tr>
<tr>
<td>Fig. 3.3f-h</td>
<td>1600</td>
<td>800</td>
<td>0.1</td>
<td>0.03</td>
<td>1ms</td>
<td>200s</td>
<td>0</td>
<td>100ms</td>
<td>100ms</td>
<td>(\lambda_s^{-1}s)</td>
</tr>
<tr>
<td>Fig. 3.4d</td>
<td>300</td>
<td>300</td>
<td>50</td>
<td>0.5</td>
<td>10ms</td>
<td>1000s</td>
<td>0</td>
<td>1s</td>
<td>0.5s</td>
<td>(\lambda_s-0.02s)</td>
</tr>
<tr>
<td>Fig. 3.5</td>
<td>500</td>
<td>300</td>
<td>0.1</td>
<td>0.03</td>
<td>1ms</td>
<td>1000s</td>
<td>(\frac{20}{N^2})</td>
<td>100ms</td>
<td>50ms</td>
<td>(\lambda_s-10s)</td>
</tr>
</tbody>
</table>

**Table 3.3:** Parameters used in the different figures for simulations of networks with nonlinear dendrites. The parameter \(a = \lambda_s - \lambda_x\) is given in terms of \(\lambda_s\) and \(\lambda_x\).

Equation (3.17) is a discrete approximation to Equation (3.25). In our task, Equation (3.24) becomes

\[
\dot{\phi}(t) = \omega(t)
\]

\[
\dot{\omega}(t) = -\omega_0^2 \sin(\phi(t)) - c \omega_0 \omega(t) + \xi(t) + u(t)
\]

and \(U(x(t)) = -y(t) = \cos(\phi(t))\).

3.5.5 Figure details

The parameters of the different simulations are given in Table 3.2 for simulations using saturating synapses and in Table 3.3 for simulations using nonlinear dendrites. Further parameters and details about the figures and simulations are given in the following paragraphs.

If not mentioned otherwise, for all simulations we use \(g = 1.5\frac{1}{s}, p = 0.1, \tilde{\nu}^f = 1\frac{1}{s}, \tilde{\nu}^i = 1\frac{1}{s}, \Delta t = 0.01s, \gamma = \theta\) and \(\sigma_\eta = 0\frac{1}{\sqrt{s}}\). We note that for simulations with saturating synapses, we model the slow synaptic currents to possess synaptic time constants of 100ms (cf., e.g., [HVG14, ZAG15]). We usually use the same value for the slow synapses in networks with nonlinear dendrites. Upon rescaling time, these networks can be interpreted as networks with faster time constants, which learn faster target dynamics. Since the spike rates scale likewise, we have to consider larger networks to generate rates in the biologically plausible range (cf. Fig. 3.11 in the supplement 3.6).
3. Learning universal computations with spikes

Figure 3.3
Figure 3.3b,c: The PCSN has non-linear dendrites. The target signal is a sine with period $4\pi$ s and amplitude 2 (normalized to one in the figure). During recall, the neurons of the PCSN spike with mean rate 30.2Hz.

Figure 3.3d: The PCSN has saturating synapses. The target signal is a saw tooth pattern with period 2s and amplitude 10 (normalized to one in the figure). We used an Euler scheme here. The mean spike rate is 226Hz.

Figure 3.3e: The task is performed by a PCSN with non-linear dendrites and by a PCSN with saturating synapses. The target signal is $\sin(t^{0.5}) + \cos(t^{1.5})$. The mean spike rate is 77.8Hz for saturating synapses and 21.3Hz for non-linear dendrites.

Figure 3.3f-h: The PCSN has nonlinear dendrites. As teacher we use the standard Lorenz system
\[
\begin{align*}
\dot{x}(t) &= \sigma (y(t) - x(t)) \\
\dot{y}(t) &= x(t)(\rho - z(t)) - y(t) \\
\dot{z}(t) &= x(t)y(t) - \beta z(t)
\end{align*}
\]

with parameters $\sigma = 10$, $\rho = 28$, $\beta = 8/3$; we set the dimensionless temporal unit to 0.2s and scale the dynamical variables by a factor of 0.1. Panels (f,g) show a recall phase of 400s, panel (h) shows points from a simulation of 4000s. Panel (f) only shows every 10th data point, panel (g) shows every data point. The mean spike rate is 432Hz.

Figure 3.4
Figure 3.4a-c: We quantified the memory capacity of a CSN with saturating synapses. The network has a sparse connectivity matrix $A$ without autapses. We applied white noise with $\sigma_\eta = 0.001 \frac{1}{\sqrt{s}}$. The input is a Gaussian bell curve with $\sigma = 0.2s$ and integral 10s (height normalized to one in the figure). The target is a Gaussian bell curve with $\sigma = 1s$ and integral 1s (height normalized to one in the figure). The target is presented several seconds after the input. Trials consisting of inputs and subsequent desired outputs are generated at random times with exponential inter-trial-interval distribution with time constant 10s and a refractory time of 100s. Training time is $T_t = 800s$, i.e. the network is trained with about 6 to 8 trials. Testing has the same duration with a similar number of trials. There is no feedback introduced by initialization or by learning, so the memory effect is purely inherent to the random network. We compute the quality of the desired output generation as the root mean squared (RMS) error between the generated and the desired response, normalized by the number of test trials. As reference, we set the error of the “extinguished” network, which does not generate any reaction to the input, to 1. Lower panels of Fig. 3.4a-c display medians and quartiles taken over 50 task repetitions. The sweep was done for time-delays $2-20s$ in steps of 0.5 $s$. 

74
3.5. Methods

Figure 3.4d: The PCSN has nonlinear dendrites. For this task a constant input of $I^\text{const} = b$ is added to the network with the elements of the vector $b$ chosen uniformly from $[0^{1\over 3}, 250^{1\over 3}]$ to introduce inhomogeneity. Four different inputs are fed into the network, two continuous $f_{1/2}^c$ and two pulsed input channels $f_{1/2}^p$. The continuous inputs are created by convolving white noise twice with an exponential kernel $e^{-t^{1\over 3}}$ (equivalent to convolving once with an alpha function) during training and $e^{-t^{1\over 10}}$ during testing. The continuous input signals are normalized to have mean 0 and standard deviation 0.5. The pulsed instruction input is created by the convolution of a Poisson spike train with an exponential kernel $e^{-t^{1\over 3}}$. The rate of the delta pulses during training is 0.041s. During testing we choose a slower rate of 0.011s for a clearer presentation. In the rare case when two pulses overlap such that the pulsed signal exceeds an absolute value of 1.01 times the maximal pulse height of one, we shift the pulse by the minimal required amount of time to achieve a sum of the pulses below or equal to 1.01. We use weights $\tilde{w}^{i,p} = 100^{1\over 3}$ for the pulsed inputs, $\tilde{w}^{i,c} = 250^{1\over 3}$ for the continuous inputs and $\tilde{w}^f = 250^{1\over 3}$ for the feedback; $g = 75^{1\over 3}$. The recurrent weights of the network are trained with respect to the memory target $F_m(t)$. This target is $+1$ if the last instruction pulse came from $f_{1}^p$ and it is $-1$ if the last pulse came from $f_{2}^p$. During switching the target follows the integral of the input pulse. The corresponding readout is $z_m$. The second readout $z_c$ is trained to output the absolute value of the difference of the two continuous inputs, if the last instruction pulse came from $f_{1}^p$, and to output their sum, if the last instruction pulse came from $f_{2}^p$. The specific analytical form of this target is $F_c(t) = \left|f_{1}^c(t) - f_{2}^c(t)\right| (F_m(t) + 1)/2 - (f_{1}^c(t) + f_{2}^c(t))(F_m(t) - 1)/2$. The mean spike rate is 5.53Hz.

Figure 3.5

Since we have white noise as input we use the Euler-Maruyama scheme in all differential equations. The PCSN has nonlinear dendrites. Non-plastic coupling strengths are $\tilde{w}^{f,y} = 100^{1\over 3}$ for the feedback of the y-coordinate, $\tilde{w}^{f,x} = 100^{1\over 3}$ for the feedback of the x-coordinate, $\tilde{w}^{f,\omega} = 20^{1\over 3}$ for the feedback of the angular velocity and $\tilde{w}^i = 2^{1\over 7}$ for the input. We introduce an additional random constant bias term into the nonlinearity to increase inhomogeneity between the neurons: The nonlinearity is $\tanh\left(b_j + \sum_{m=1}^{N} W_{njm}r_m(t)\right)$ where $b_j$ is drawn from a Gaussian distribution with standard deviation 0.01. The integration time $\delta$ is 0.1s. During the control/testing phase, every $\Delta = 0.01s$, $M = 200$ samples of length $T_r = 1s$ are created, the cost function is weighted with $\lambda_c = 0.01^{1\over 3}$. The mean spike rate is 146Hz.
3. Learning universal computations with spikes

3.6 Supplement: Learning universal computations with spikes

3.6.1 Networks with nonlinear dendrites

As stated in the main text, we can generalize Equation (3.3) by introducing fast connections that generate discontinuities in postsynaptic neurons when a neuron spikes. We may then require that only a lower, say $J-$, dimensional combination $x(t)$ of the $N-$dimensional vectors $V(t)$ and $r(t)$ is continuous,

$$x(t) = LV(t) + \tilde{\Gamma}r(t),$$

(3.26)

where $L$ and $\tilde{\Gamma}$ are $J \times N$ matrices. (For clarity of presentation we will use vector/matrix notation instead of components throughout the present section.) The benefit of this approach is that the spike trains of a larger population of neurons contribute to each $x_n$, such that a modified analogue to Equation (3.9),

$$x(t) \approx \Gamma r(t),$$

(3.27)

with a $J \times N$ matrix $\Gamma$ can hold even if the spike rates of individual neurons are low, i.e. if we make use of population/distributed coding. The matrix $L$ is fixed (except for degenerate cases) as soon as the matrix $\tilde{\Gamma}$ and the fast changes in $V$ are fixed. However, it is not a priori clear how to choose the latter two; we need to employ some optimization scheme to ensure both a good approximation Equation (3.27) and a low firing rate.

For this, we start anew, and in contrast to the previous section with the dynamics for $x(t)$. From these we will derive spiking dynamics approximating the $x(t)$. We begin with a general $J$-dimensional nonlinear dynamical system yielding $x(t)$,

$$\dot{x}(t) = f(x(t)) + c(t),$$

(3.28)

where $f(x)$ and $c(t)$ are column vectors of functions $f_j(x_1, ..., x_N)$ and external inputs $c_j(t)$, respectively. We will generalize an approach introduced in refs. [BD11, BBMD12, BMD13] to nonlinear systems and derive spiking dynamics that optimally (see below) approximate $x(t)$ satisfying Equation (3.28). The approach will yield Equation (3.26) with a specific $L$ as by-product. We will find that the dynamics of individual neurons depend on the $f_j$ and we will specify these functions such that the neural dynamics are biologically plausible and suitable for universal computation.

We choose the momentary error or cost function

$$E(t) = (x(t) - \Gamma r(t))^2 + \mu r^2(t)$$

(3.29)

to be minimized at each time $t$. The first term in $E(t)$ induces the approximation Equation (3.27), the second term fosters a low spike rate with spiking distributed
over all neurons. The error function respects causality as it depends implicitly via \( x(t) \) and \( r(t) \) on the past and restrains the dynamics at the current time \( t \) only. Minimizing \( E(t) \) at \( t \) means that a spike should be sent by neuron \( n \) if \( E(t) \) decreases due to this spike. Comparing \( E_n(t) \) (spike sending at time \( t \) by neuron \( n \)) with \( E_0(t) \) (no spike sending) yields

\[
E_n(t) < E_0(t) \tag{3.30}
\]

\[
(x(t) - \Gamma r(t) - \Gamma \hat{e}_n)^2 + \mu (r(t) + \hat{e}_n)^2 < (x(t) - \Gamma r(t))^2 + \mu r^2(t) \tag{3.31}
\]

\[
\Gamma_n \cdot (x(t) - \Gamma r(t)) - \mu r_n(t) > \frac{\Gamma_n^2 + \mu}{2}, \tag{3.32}
\]

where \( \hat{e}_n \) denotes the \( n \)-th unit vector, \( \hat{e}_n = (0, ..., 1, 0, ...)^T \) (with a 1 in the \( n \)-th row), and \( \Gamma_n \) is the \( n \)-th column (vector) of the matrix \( \Gamma \), \( \Gamma_n = \Gamma \hat{e}_n \). To obtain the familiar condition \( V_n(t) > \theta_n \) for neuron \( n \) to spike, the variable left hand side of Equation (3.32) may be interpreted as membrane potential,

\[
V_n(t) = \Gamma_n \cdot (x(t) - \Gamma r(t)) - \mu r_n(t), \tag{3.33}
\]

\[
V(t) = \Gamma^T (x(t) - \Gamma r(t)) - \mu r(t), \tag{3.34}
\]

the right hand side as threshold

\[
\theta_n = \frac{\Gamma_n^2 + \mu}{2}. \tag{3.35}
\]

We note that we can multiply both sides of the Equation by a factor and add constant terms, these change the scale of the potential, and shift the resting membrane potential, the reset and the threshold. Equation (3.34) yields Equation (3.26) with the pseudo-inverse of \( \Gamma^T \), \( \mathbf{L} = (\Gamma \Gamma^T)^{-1} \Gamma \), and \( \tilde{\Gamma} = \Gamma + \mu \mathbf{L} \).

We can now derive the sub-threshold dynamical Equations for \( V(t) \) from those for \( x(t) \) and \( r(t) \):

\[
\dot{V}(t) = \Gamma^T (x(t) - \Gamma r(t)) - \mu r(t) \tag{3.36}
\]

\[
= \Gamma^T f(x(t)) - (\Gamma^T \Gamma + \mu \mathbf{1})(s(t) - \lambda_s r(t)) + \Gamma^T \mathbf{c}(t), \tag{3.37}
\]

where \( \mathbf{1} \) denotes the \( N \times N \) identity matrix. Assuming that the minimization of Equation (3.29) yields small \( E(t) \), we may eliminate the dependence on \( x(t) \) using Equation (3.27),

\[
\dot{V}(t) \approx \Gamma^T f(\Gamma r(t)) - (\Gamma^T \Gamma + \mu \mathbf{1})(s(t) - \lambda_s r(t)) + \Gamma^T \mathbf{c}(t). \tag{3.38}
\]

Finally, biological realism and increased stability of numerical simulations indicate that an additional leak term \( -\lambda_V V \) should be introduced

\[
\dot{V}(t) = -\lambda_V V(t) + \Gamma^T f(\Gamma r(t)) - (\Gamma^T \Gamma + \mu \mathbf{1})(s(t) - \lambda_s r(t)) + \Gamma^T \mathbf{c}(t). \tag{3.39}
\]
3. Learning universal computations with spikes

We now choose the \( f_j \) as

\[
f_j(x_1, \ldots, x_J) = -\lambda_x x_j + \sum_{i=1}^J A_{ji} \tanh(x_i),
\]

(3.40)

such that

\[
\dot{V}(t) = -\lambda V(t) V(t) + \Gamma^T A \tanh(\Gamma r(t)) - (\Gamma^T \Gamma + \mu) s(t) + (a \Gamma^T \Gamma + \mu \lambda_x) r(t) + \Gamma^T c(t),
\]

(3.41)

where \( a = \lambda_x - \lambda_x \). This yields a spiking neural network that is suitable for universal computation: Its dynamics can be decoded via Equation (3.26) (or Equation (3.27)) to resemble those of a \( J \)-dimensional dynamical system of the form

\[
\dot{x}(t) = -\lambda_x x(t) + A \tanh(x(t)) + c(t).
\]

(3.42)

Systems of the form Equation (3.42) are known to be suitable for universal computation [JH04, SA09, LJS12], in particular for appropriate \( A \) they can maintain longer-term fading memory. Since the \( x \) are dynamical quantities linearly derived from the underlying spiking network, already the underlying spiking network is suitable for computations.

Furthermore, the structure of Equation (3.41) allows for a straightforward interpretation in biological terms: The response of neuron \( n \)'s soma to slow input to its \( J \) dendrites is modeled by the term \( \Gamma_n \cdot (A \tanh(\Gamma r(t))) \). The inputs have non-negligible synaptic time constant (cf. \( r(t) \)), they are linearly summed and thereafter subjected to a dendritic sublinearity (\( \tanh \)). The coupling strength of a synaptic connection from neuron \( m \) to the \( j \)th dendrite of neuron \( n \) is given by \( \Gamma_{jm} \), the coupling strength from the \( j \)th dendrite of neuron \( n \) to its soma is \( (\Gamma^T A)_{nj} \). Further fast and slow inputs arriving near the soma (and thus not subject to a dendritic non-linearity) are incorporated by the terms \( -\Gamma^T \Gamma \mu_1 s(t) \) and \( (a \Gamma^T \Gamma + \mu \lambda_x) r(t) \). Their impact is characterized by the product \( \Gamma^T \Gamma \) of the decoding matrix with itself and the comparably small weight \( \mu \) of the spike frequency penalty term, the positive diagonal terms incorporate the reset of the neurons after a spike and a slower recovery.

3.6.2 A sufficient condition for the echo state property of the dynamics Equation (3.6)

When does

\[
\dot{x}(t) = -\lambda_v [x(t)]_\mu - \lambda_x [x(t)]_+ + A \tanh([x(t)]_+) + I(t)
\]

(Equation (3.6) of the main text) possess the echo state property? Dynamics have this property if after sufficiently long time any initial conditions are washed out and the state of the system is completely determined by the input. This is definitely
the case if the distance between trajectories decays at least exponentially with a rate independent of the input [Jae01]. We will prove the latter for our dynamics Equation (3.43). For this, we will consider the difference $\Delta(t) = x_1(t) - x_2(t)$ and the Euclidean distance $\|\Delta(t)\|$ of two trajectories that satisfy Equation (3.43) and have different initial conditions $x_1(0), x_2(0)$ but the same input $I(t)$. We will show that under the condition $\|A\| < \min(\lambda_V, \lambda_x)$, with $\|A\|$ being the spectral norm (the largest singular value) of $A$, an inequality $\|\Delta(t)\| \leq -\epsilon \|\Delta(t)\|$ holds for some $\epsilon > 0$ (as usual the dot denotes the temporal derivative of the entire expression below, here $\|\Delta(t)\|$).

We start with the expression $\frac{1}{2}\|\Delta(t)\|^2 = \Delta(t) \dot{\Delta}(t)$ and replace the right hand side using $\Delta(t) = x_1(t) - x_2(t)$ and Equation (3.43), which leads to

$$\frac{1}{2}\|\Delta(t)\|^2 = (x_1 - x_2)(-\lambda_V([x_1(t)_] - [x_2(t)_] - \lambda_x([x_1(t)_] - [x_2(t)_] ) ) + (x_1 - x_2)(A(tanh([x_1(t)_] - tanh([x_2(t)_] )) + I(t) - I(t)).$$

(3.44)

To proceed we use the three inequalities $xy \leq \|x\| \|y\|$, $\|Ax\| \leq \|A\| \|x\|$ and $\|\tanh(x) - \tanh(y)\| \leq \|x - y\|$, which allow to estimate

$$(x_1 - x_2)A(\tanh([x_1(t)_] - \tanh([x_2(t)_] )) \leq \|x_1 - x_2\| \|A\| \|[x_1(t)_] - [x_2(t)_] \|.$$  

(3.45)

We now simplify the right hand side of the inequality further. For this we use that for every pair of real valued vectors $x$ and $y$ we have $[x]_+ [y]_+ \geq 0$, $[x]_- [y]_+ \leq 0$ and $[x]_+ [x]_- = 0$, since every element of $[x]_+$ is larger/equal zero while every element of $[x]_-$ is smaller/equal zero, and elements which are nonzero in $[x]_+$ are zero in $[x]_-$ and vice versa. With this we get

$$\|x_1 - x_2\|^2 = \|[x_1(t)_] - [x_2(t)_]\|^2 + \|[x_1(t)_] - [x_2(t)_]\|^2 - 2[x_1(t)_] [x_2(t)_] - 2[x_2(t)_] [x_1(t)_] \geq \|[x_1(t)_] - [x_2(t)_]\|^2,$$

(3.46)

since $-[x_1(t)_] [x_2(t)_] - [x_2(t)_] [x_1(t)_] \geq 0$. The result can be used to bound the right hand side of Equation (3.45) by a simpler expression,

$$(x_1 - x_2)A(\tanh([x_1(t)_] - \tanh([x_2(t)_] )) \leq \|x_1 - x_2\|^2 \|A\|.$$  

(3.47)

Now we assume $\|A\| < \min(\lambda_V, \lambda_x)$ such that we can write $\lambda_V = \epsilon_V + \|A\|$ and $\lambda_x = \epsilon_x + \|A\|$ with $\epsilon_V > 0$ and $\epsilon_x > 0$. Using this in Equation (3.44) yields
3. Learning universal computations with spikes

\[
\frac{1}{2} \|\Delta(t)\|^2 = -\epsilon \|V\| \|x_1 - x_2\| (x_1(t)_{-} - x_2(t)_{-})
\]

\[
-\epsilon \|\|x_1 - x_2\||x_1 - x_2\|(x_1(t)_{+} - x_2(t)_{+})
\]

\[
+ (x_1 - x_2)A(\tanh([x_1(t)]_{+}) - \tanh([x_2(t)]_{+}))
\]

\[
= -\epsilon \|V\| (x_1 - x_2)[x_1(t)_{-} - x_2(t)_{-}] - \epsilon \|x_1 - x_2\| (x_1(t)_{+} - x_2(t)_{+})
\]

\[
- \|A\| \|x_1 - x_2\|^2 + (x_1 - x_2)A(\tanh([x_1(t)]_{+}) - \tanh([x_2(t)]_{+}))
\]

and together with Equation (3.47)

\[
\frac{1}{2} \|\Delta(t)\|^2 \leq -\epsilon \|V\| (x_1 - x_2)[x_1(t)_{-} - x_2(t)_{-}] - \epsilon \|x_1 - x_2\| (x_1(t)_{+} - x_2(t)_{+})
\]

\[
- \|A\| \|x_1 - x_2\|^2 + \|x_1 - x_2\|^2 \|A\|
\]

\[
= -\epsilon \|V\| (x_1 - x_2)[x_1(t)_{-} - x_2(t)_{-}] - \epsilon \|x_1 - x_2\| (x_1(t)_{+} - x_2(t)_{+})
\]

(3.49)

Both terms on the right hand side are smaller or equal to zero,

\[
(x_1 - x_2)([x_1(t)]_{-} - [x_2(t)]_{-}) =
\]

\[
([x_1(t)]_{+} - [x_2(t)]_{-}) - [x_1(t)]_{+} [x_2(t)]_{+} [x_1(t)]_{-} \geq 0.
\]

(3.50)

We can therefore set \( \epsilon = \min(\epsilon_V, \epsilon_X) > 0 \) and simplify

\[
\frac{1}{2} \|\Delta(t)\|^2 \leq -\epsilon (x_1 - x_2)([x_1(t)]_{-} - [x_2(t)]_{-}) - \epsilon (x_1 - x_2)([x_1(t)]_{+} - [x_2(t)]_{+})
\]

\[
= -\epsilon \|\Delta(t)\|^2,
\]

(3.51)

which is equivalent to

\[
\|\Delta(t)\| \leq -\epsilon \|\Delta(t)\|.
\]

(3.52)

The distance between different trajectories thus decreases at least exponentially fast with rate \( \epsilon \), for any input. We may conclude that \( \|A\| < \min(\lambda_V, \lambda_X) \) provides a sufficient condition for the system to possess the echo-state property.

### 3.6.3 Comparison of PCSNs with Poisson coding learning networks

In the following, we compare the performance of PCSNs and Poisson coding learning networks. To enable a direct comparison, we use PCSNs with saturating synapses
Figure 3.6: Comparison of PCSNs and Poisson coding learning networks. Error of PCSNs and Poisson coding learning networks with different spike rates after learning continuous dynamics. The panel shows the median error to the saw tooth target pattern (cf. Fig. 3.3d) during testing, in equidistant bins of the network spike rate (shaded: intervals between first and third quartile). The PCSN with its deterministic spike code reaches the same error level as the networks with simple Poisson coding with almost two orders of magnitude fewer spikes.
and Poisson coding networks of the same size and with the same learning rule for the recurrent synapses such that in the high-rate limit, both network types become equivalent to the same continuous networks. As a specific task for the comparison, we choose learning of a saw tooth-like signal as displayed in Fig. 3.3d. We find that both networks perform better for higher rates. However, due to their deterministic, precise spike code, the PCSNs achieve the same error levels with almost two orders of magnitude smaller rates, cf. Fig. 3.6. This is generally a consequence of the fact that the population coding error in precisely spiking networks is much smaller than in Poisson coding networks, ref. [BMD13] shows it to be proportional to $1/N$ (where $N$ is the number of neurons in the network), while a simple Poisson population code has precision $1/\sqrt{\pi}$. However, in our PCSNs we have additional learning whose consequences on the precision of the output signal are not easy to determine.

The Poisson models are setup as follows: We start with the continuous target dynamics Equation (3.6) and for simplicity consider $\lambda_x = \lambda_V$ and $\gamma = \theta$, i.e.

$$\dot{x}_n(t) = -\lambda_V x_n(t) + \sum_{m=1}^{N} A_{nm} \tanh([x_m(t)]_+) + I_{e,n}(t).$$

(3.53)

The state of the corresponding Poisson unit $n$ shall be characterized by $u_n(t)$; we aim at $u_n(t) \approx x_n(t)$ for high spike rates. For the spike generation, we orient at standard models (e.g. [GK01]) and at keeping the dynamical Equations simple.

As Poisson model 1, we use networks of units with threshold and nonlinear saturation, specifically unit $n$ has the rate

$$v_n(t) = s_0 \tanh([u_n(t)]_+).$$

(3.54)

The constant $s_0$ allows to modulate the rate without changing the dynamics of $u$. Given $v_n(t)$, the unit generates an inhomogeneous Poisson spike train $s_n(t)$ (cf. Equation (3.1)) with this rate. The spike train in turn generates postsynaptic inputs with decay time constant $\lambda_s$, as given in Equation (3.2). When rescaled with $\lambda_s/s_0$, the postsynaptic inputs satisfy for large $s_0$

$$\frac{\lambda_s}{s_0} r_n(t) \approx \frac{v_n(t)}{s_0} = \tanh([u_n(t)]_+).$$

(3.55)

A network with dynamics

$$\dot{u}_n(t) = -\lambda_V u_n(t) + \sum_{m=1}^{N} A_{nm} \frac{\lambda_s}{s_0} r_m(t) + I_{e,n}(t)$$

(3.56)

then approximates the continuous dynamics Equation (3.53).

As Poisson model 2, we use networks of linear threshold units,

$$v_n(t) = s_0 [u_n(t)]_+.$$
They yield for large $s_0$

$$\frac{\lambda_s}{s_0} r_n(t) \approx [u_n(t)]_+ \tag{3.57}$$

and

$$\dot{u}_n(t) = -\lambda_V u_n(t) + \sum_{m=1}^{N} A_{nm} \tanh \left( \frac{\lambda_s}{s_0} r_m(t) \right) + I_{e,n}(t) \tag{3.58}$$

for the network dynamics approximating Equation (3.53). We note that this model also satisfies a decoding Equation analogous to Equation (3.9), $\frac{\lambda_s}{s_0} r_n(t) \approx [x_n(t)]_+$.

We change the threshold $\theta$ (PCSNs) and the base rates $s_0$ (Poisson networks) to generate networks with different rates. For PCSNs the remaining parameters are adapted such that the corresponding continuous network is also given by Equation (3.53). For each parameter value we train 75 networks with different random topology and different initial conditions. We thereafter compute the actually generated average spike rates within log-scale equidistant bins. Further, we compute the root mean squared (RMS) error between the desired signal and the signal generated during testing. Fig. 3.6 displays the median and the first and third quartiles of the error versus the average rate in double logarithmic scale.

### 3.6.4 Robustness against noise

PCSN learning is robust against noise. Fig. 3.7 shows this by example of the learning of the saw tooth pattern (cf. Figs. 3.3d, 3.6), both for PCSNs with saturating synapses and nonlinear dendrites. In each time step of the Euler-Maruyama integration, Gaussian noise is added. The noise level is given in terms of the standard deviation generated by a purely noise-driven subthreshold membrane potential (Ornstein-Uhlenbeck process) with membrane time constant $\lambda_V$, in multiples of the threshold (which equals $\theta/2$ in the case of saturating synapses and $\theta$ in the case of non-linear dendrites), i.e. noise-level := $\frac{\sigma_n}{\sqrt{2\lambda_V}} \sqrt{(1-e^{-2})}$. The error is determined as RMS error between the desired signal and the signal generated during testing.

### 3.6.5 Robustness of PCSNs with nonlinear dendrites against structural perturbations

In CSNs with nonlinear dendrites, the optimal strengths of the couplings from other neurons to the nonlinear dendrites and the fast couplings are independent of the parameters of the encoded nonlinear dynamical system Equation (3.42). Since deviations from the optimal values in these couplings in general do not imply a simple change in the encoded dynamics but impair the coding scheme, we here investigate robustness of PCSN-learning against them.

The optimal coupling strength from neuron $m$ to the $j$th nonlinear dendrite of neuron $n$ is $W_{njm} = \Gamma_{jm}$. Here we test the robustness of the learning scheme against
3. Learning universal computations with spikes

Figure 3.7: Robustness of PCSNs against noise. The figure shows the median RMS error between the output of PCSNs and the saw tooth target pattern during testing, versus the noise level (shaded: intervals between first and third quartile). The noise level is given in terms of the standard deviation generated by a purely noise-driven subthreshold membrane potential (Ornstein-Uhlenbeck process) with membrane time constant \( \lambda_V \), in multiples of the threshold. The insets display the testing phase for two examples using the setup with nonlinear dendrites (crosses in the main plot denote the corresponding noise and error levels).
Figure 3.8: Robustness of PCSNs against structural perturbations of dendritic coupling. The figure shows performance of native PCSNs (networks without precise reset, green) and for PCSNs where neurons are reset to a fixed membrane potential after spike generation (networks with precise reset, blue). Displayed is the median RMS error between the output of PCSNs and the saw tooth target pattern during testing versus the standard deviation $W_{\text{pert}}$ of a multiplicative Gaussian perturbation in the connectivity (shaded: intervals between first and third quartile). The insets display the testing phase for two examples using the setup with precise reset (crosses in the main plot denote the corresponding $W_{\text{pert}}$ and error levels). While the native model does not show successful learning for perturbations larger than 2% of the connection weights, networks with precise reset generate a recognizable sawtooth pattern as learned output even for perturbations of 20% (inset at lower right). The error generated by networks with precise reset increases gradually with perturbation strength, while there is an abrupt change for the native networks.
3. Learning universal computations with spikes

Figure 3.9: Robustness of PCSNs against reduction of fast couplings. The panel shows the median RMS error between the output of PCSNs and the saw tooth target pattern during testing versus the size $s_{\text{pert}}$ of the multiplicative reduction of the fast connections (shaded: intervals between first and third quartile). For $s_{\text{pert}} < 10\%$ the error increases only slightly with increasing $s_{\text{pert}}$. A further increase of $s_{\text{pert}}$ leads to a strong, rapid increase in the error and the PCSN is soon not able to learn the pattern anymore.
deviations from the optimal couplings. We find that PCSN learning is robust, if we modify the neuron model to have a “precise reset”, i.e. the reset is always to the fixed value $-\theta$ ($-2\theta$ below threshold $\theta$), even if fast excitation to a suprathreshold potential caused the spike (Fig. 3.8). In the native model the reset has fixed size $-2\theta$ such that the membrane potential would be reset to a value larger than $-\theta$ after a suprathreshold excitation. We note that the reset to a fixed value may also be biologically more plausible than a reset of fixed size.

Fig. 3.8 shows this by example of the learning of the saw tooth pattern (cf. Figs. 3.3d, 3.6). The $W_{njm}$ are perturbed proportionally to the strength of their optimal values, the perturbed couplings are given by $W_{njm} = \Gamma_{jm} \left( 1 + W_{pert} \Xi_{njm} \right)$, where the $\Xi_{njm}$ are independently drawn from a Gaussian distribution with mean zero and variance one. We adopt the interpretation of the PCSNs as networks with plastic recurrent connections: The outputs, the output weight updates, the inverse correlation matrices and the updates of the dendrite-to-soma weights $D_{nj}$ are computed using Equations (3.14), (3.18), (3.19) and (3.22) with unperturbed readouts $\tilde{r}_j(t) = \tanh \left( \sum_{m=1}^{N} \Gamma_{jm} r_m(t) \right)$. We note that updating the $D_{nj}$ is not equivalent to a static feedback of the updated overall network readout anymore, since the latter does not contain the perturbed $W_{njm}$.

Our findings raise the question why the networks with precise reset are much more stable to perturbations in the dendrites. We find that the instability in the simulations of the conventional model is due to an explosion of the spike rates, such that every neuron spikes once at every simulated time step. This is due to the “ping pong effect” already described in ref. [BMD13]: Neurons that spike due to excitation from fast connections generate further suprathreshold excitation and in the end all neurons have spiked within a single step. In [BMD13] this problem is solved using a higher value of $\mu$. We find for our simulations with perturbed $W_{njm}$ that a fine tuning of $\mu$ is required to prevent a breakdown of the learning. In contrast, the precise reset solves this problem robustly. This is a consequence of the fact that on the one hand the precise reset yields a membrane potential that is further away from the threshold and thus reduces the chance of re-excitation of a neuron that has recently spiked ($\mu$ has an in principle similar effect). On the other hand, the difference from the theory arises only after suprathreshold excitation, the change in the precise spiking dynamics is small and the spike coding scheme is preserved.

The optimal strength of a fast coupling from neuron $m$ to neuron $n$ is $U_{nn} = \sum_{j=1}^{J} \Gamma_{jn} \Gamma_{jm} + \mu \delta_{nm}$ (which is independent of $N$ for fixed neuron threshold). In “balanced state” irregular spiking networks, such recurrent connections may be expected to be much smaller, since they scale with $1/\sqrt{N}$ for fixed neuron thresholds [RdB+10]. We therefore test the dependence of the PCSNs on these connections by a multiplicative weakening by a factor $1 - s_{pert}$, $U_{pert}^{nm} = (1 - s_{pert}) U_{nm}$ for $n \neq m$ (resets are kept $U_{pert}^{nn} = U_{nn}$). Fig. 3.9 shows that PCSNs are robust against this:
The learning capabilities are conserved, if $s_{pert} < 10\%$. The figure also indicates that the fast connections are important for PCSN functionality, since the learning abilities are quickly lost as $s_{pert}$ increases beyond this range.

3.6.6 Comparison of network sizes and the spike rate-network size trade-off

We illustrate that for PCSNs, network sizes comparable to those of continuous rate networks solving the same task with FORCE learning can be sufficient. Further, we show that there is a trade-off between network size and spike rate of individual neurons. As example we use the “camel's hump” task (cf. Fig. 3.3e). Fig. 3.11a shows that continuous networks Equation (3.6) can learn the signal well for about $N > 40$, we use $N = 50$ as a reference. We compare with PCSNs with nonlinear dendrites that encode a system Equation (3.6) with the same parameters (in particular $J = 50$), and are trained to solve the same task. We compare the RMS error and spiking frequencies for different PCSN sizes $N$ and $\gamma_s$, which regulates the threshold of the neurons (cf. Methods and Equation (3.35)). Fig. 3.11b,c shows the trade-off between the number of neurons in the PCSNs and their individual spiking frequency: For $\gamma_s \approx 0.1 - 0.15$ only the networks with $N = 200$ and $N = 400$ learn the task reliably (panel (b)), the neurons adopt a mean spike rate of about or smaller 100Hz. For sufficiently small $\gamma_s$ also smaller networks learn the task well, but all networks generate a higher spike frequency.

3.6.7 Error evolution for longer times

We do not observe lasting changes of the error in long term simulations. For periodic signals, there is an inevitable phase shift. It originates from the small error between the period of the desired and the learned signal; this error accumulates over time. Apart from that, tested features such as the deviation from the desired dynamics (RMS error) and the spike frequency are remarkably constant. Fig. 3.12a,b illustrates and quantifies this for the camel’s hump task (Fig. 3.3e). Fig. 3.12a displays the continued desired dynamics and the occurring phase shift for longer recall durations. Fig. 3.12b shows that the RMS error is stationary, approximately constant over time, if one corrects for the shift. For the Lorenz attractor (Fig. 3.3f-h), the teacher and student trajectories quickly depart from each other after the end of learning. The spiking network nevertheless continues to generate dynamics that after decoding agree with the dynamics of a Lorenz system. Fig. 3.12c shows this for longer times. As a quantitative check, the tent map Fig. 3.3h relating subsequent local maxima in the $z$-coordinate shows the good agreement with the tent-map generated by the teacher dynamics also for long times. In the displayed simulation, the Lorenz dynamics deviate three times from the desired dynamics, which leads to the six outliers in Fig. 3.3h. However, the dynamics return every time to the desired
Figure 3.10: Memory duration in CSNs with different recurrent coupling strengths. Supporting figure to Fig. 3.4a-c, displaying a direct comparison between multiple error vs. reaction delay traces. A disconnected network, $g \lambda_x^{-1} = 0$, has comparably short memory. Increase of connection strength leads to an increase of memory duration (cf. the trace for $g \lambda_x^{-1} = 0.8$). Memory is most persistent around $g \lambda_x^{-1} = 1.6$, and decreases for larger coupling strengths, as expected for systems where the dynamics become more and more chaotic ($g \lambda_x^{-1} = 2$, $g \lambda_x^{-1} = 8$).
3. Learning universal computations with spikes

Figure 3.11: Network size and PCSN spiking rate trade-off. (a): RMS errors of continuous rate networks of different size, after training the camel’s hump task (cf. Fig. 3.3e). The networks generate the pattern well for network sizes larger approximately $N = 40$. (b,c): RMS errors of PCSNs with nonlinear dendrites, after training the camel’s hump task. The panels display the trade-off between network size and spike frequency: For $N = 200$, a small error can be achieved with a single neuron spike rate of 100Hz (cf. RMS error at $\gamma_s \approx 0.1$ displayed in (b) and rate at $\gamma_s \approx 0.1$ displayed in (c)). Networks with $N = 400$ need only a spike rate of 10Hz. Smaller networks (e.g. $N = 70$) need higher spike rates but reach the same error levels. (We note that for small networks we observe a nonmonotonic dependence of the spike rate on the network size for constant $\gamma_s$.) (d): Example dynamics. The PCSN signal (red trace, $\gamma_s = 0.01$, $N = 70$, $J = 50$) approximates the target function (yellow trace) similarly well as a continuous rate neuron network of similar size (blue trace, $N = 50$).
3.6. Supplement: Learning universal computations with spikes

Figure 3.12: Long term evolution. (a): PCSN generated camels’s hump signal and target (Figs. 3.3e, 3.11) after long times: the signal is phase-shifted compared to the target but otherwise not noticeably changed. Panel (b) quantifies this observation by plotting the RMS error, corrected for a possible phase shift to the target, against time. Displayed are several learning trials with different random initial connectivity. The error is stationary, constant except for fluctuations. (c): Lorenz attractor signal of Fig. 3.3f-h, dynamics of $x, y, z$ vs. time. Also after long times, the PCSN (red trace) in general generates qualitatively the same dynamics as the target Lorenz system (yellow trace).
3. Learning universal computations with spikes

### Table 3.4: Parameters used in the different figures for simulations of networks with saturating synapses.

<table>
<thead>
<tr>
<th>Sat. syn.</th>
<th>N</th>
<th>α</th>
<th>dt</th>
<th>$T_t$</th>
<th>$\lambda_s^{-1}$</th>
<th>$\lambda_V^{-1}$</th>
<th>$V_r$</th>
<th>$\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 3.6 (PCSN)</td>
<td>50</td>
<td>0.01</td>
<td>0.001ms</td>
<td>10.5s</td>
<td>100ms</td>
<td>1s</td>
<td>0.9θ</td>
<td>swept</td>
</tr>
<tr>
<td>Fig. 3.7</td>
<td>100</td>
<td>0.1</td>
<td>0.1ms</td>
<td>30s</td>
<td>100ms</td>
<td>50ms</td>
<td>0.9θ</td>
<td>0.05</td>
</tr>
</tbody>
</table>

dynamics such that the errors just generate single outlier pairs in the tent map and the qualitative dynamics agree with those of the Lorenz system still after 4000s.

### 3.6.8 Details on the supporting figures

The parameters of the different simulations are given in Table 3.4 for simulations using saturating synapses and in Table 3.5 for simulations using nonlinear dendrites. Further parameters and details about the figures and simulations are given in the following paragraphs.

If not mentioned otherwise, for all simulations we use $g = 1.5 \frac{1}{s}$, $p = 0.1$, $\tilde{w}^f = 1 \frac{1}{s}$, $\tilde{w}^i = 1 \frac{1}{s}$, $\Delta t = 0.01s$, $\gamma = \theta$ and $\sigma_\eta = 0 \frac{1}{v^2s}$.

#### Figure 3.6

The signal has period $2s$ and amplitude $10$. The parameters of the Poisson networks are $N = 50$, $\alpha = 0.01$, $g = 1.5 \frac{1}{s}$, $T_t = 10.5s$, $\lambda_s^{-1} = 100ms$, $\lambda_V^{-1} = 1s$, $dt = 0.1/s_0$, the sparse matrix $A$ has a fraction $p = 0.1$ of nonzero entries, which are drawn from a Gaussian distribution with zero mean and variance $\frac{g^2}{pN}$. $s_0$ is swept between $50 \frac{1}{s}$ and $21544 \frac{1}{s}$ (values in the different sweeps: $s_0 = 50 \frac{1}{s}$, $100 \frac{1}{s}$, $150 \frac{1}{s}$, $200 \frac{1}{s}$, $215 \frac{1}{s}$, $464 \frac{1}{s}$, $1000 \frac{1}{s}$, $2150 \frac{1}{s}$, $4641 \frac{1}{s}$, $10000 \frac{1}{s}$, $21544 \frac{1}{s}$). The PCSN has saturating synapses. $\theta$ is swept between 0.5 and 0.01 (specific values of the sweep: $\theta = 0.5$, 0.4, 0.3, 0.2, 0.1, 0.09, 0.08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.01). We compute the RMS error between the signal and the target in the first $10.5s$ after training. The error is computed using a normalized version of the signal, where the amplitude is set to 1.

#### Figure 3.7

The signal has period $2s$ and amplitude $15$ (normalized to one in the figure). We take medians and quartiles over 50 trials. The noise level is given in terms of the standard deviation generated by a purely noise-driven subthreshold membrane potential (Ornstein-Uhlenbeck process) with membrane time constant $\lambda_V$, in multiples of the threshold (which equals $\theta/2$ in the case of saturating synapses and $\theta$ in the case of non-linear dendrites), i.e. noise-level $:= \frac{\sigma_\eta}{\sqrt{2\lambda_V}} \sqrt{1-e^{-2t}}$. Plotted are the median.
### Table 3.5: Parameters used in the different figures for simulations of networks with nonlinear dendrites. The parameter \( a = \lambda_s - \lambda_x \) is given in terms of \( \lambda_s \) and \( \lambda_x \)

<table>
<thead>
<tr>
<th>Nonlin. dendr.</th>
<th>N</th>
<th>J</th>
<th>( \alpha )</th>
<th>( \gamma_s )</th>
<th>dt</th>
<th>( T_t )</th>
<th>( \frac{1}{N^2} )</th>
<th>( \lambda_s^{-1} )</th>
<th>( \lambda_x^{-1} )</th>
<th>( a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 3.7</td>
<td>200</td>
<td>100</td>
<td>0.1</td>
<td>0.05</td>
<td>0.1ms</td>
<td>30s</td>
<td>( \frac{1}{N^2} )</td>
<td>100ms</td>
<td>50ms</td>
<td>( \lambda_s )-1Hz</td>
</tr>
<tr>
<td>Fig. 3.8</td>
<td>200</td>
<td>100</td>
<td>0.1</td>
<td>0.05</td>
<td>0.1ms</td>
<td>30s</td>
<td>( \frac{1}{N^2} )</td>
<td>100ms</td>
<td>50ms</td>
<td>( \lambda_s )-1Hz</td>
</tr>
<tr>
<td>Fig. 3.9</td>
<td>200</td>
<td>100</td>
<td>0.1</td>
<td>0.05</td>
<td>0.1ms</td>
<td>30s</td>
<td>( \frac{1}{N^2} )</td>
<td>100ms</td>
<td>50ms</td>
<td>( \lambda_s )-1Hz</td>
</tr>
<tr>
<td>Fig. 3.11b, c</td>
<td>swept</td>
<td>50</td>
<td>1</td>
<td>swept</td>
<td>0.1ms</td>
<td>100.5s</td>
<td>0</td>
<td>100ms</td>
<td>100ms</td>
<td>( \lambda_s )-1Hz</td>
</tr>
<tr>
<td>Fig. 3.11d</td>
<td>70</td>
<td>50</td>
<td>1</td>
<td>0.01</td>
<td>0.1ms</td>
<td>100.5s</td>
<td>0</td>
<td>100ms</td>
<td>100ms</td>
<td>( \lambda_s )-1Hz</td>
</tr>
<tr>
<td>Fig. 3.12a, b</td>
<td>70</td>
<td>50</td>
<td>1</td>
<td>0.01</td>
<td>0.1ms</td>
<td>100.5s</td>
<td>0</td>
<td>100ms</td>
<td>100ms</td>
<td>( \lambda_s )-1Hz</td>
</tr>
</tbody>
</table>
of the RMS error (shaded: intervals between first and third quartile) between the signal and the target in the first 30s after training. The error is computed using the normalized version of the signal, where the amplitude is set to 1. The sweep covers 101 equidistant values of $\sigma_n$ from $0 \frac{1}{\sqrt{3}}$ to $0.01 \frac{1}{\sqrt{3}}$ in the case of non-linear dendrites and 101 equidistant values of $\sigma_n$ from $0 \frac{1}{\sqrt{3}}$ to $0.2 \frac{1}{\sqrt{3}}$ in the case of saturating synapses.

**Figure 3.8**

The signal has period 2s and amplitude 10 (normalized to one in the figure). We take medians and quartiles over 50 trials. We use the Euler method to integrate the differential Equations.

Plotted are the median of the RMS error (shaded: intervals between first and third quartile) between the signal and the target in the first 30s after training. The error is computed using the normalized version of the signal, where the amplitude is set to 1. The sweep covers 80 equidistant values of $W_{\text{pert}}$ from 0 to 0.2.

For simulations that showed pathological spiking (more than 200 spikes per time step in the numerical simulation) we assigned an infinite error.

**Figure 3.9**

The signal has period 2s and amplitude 10 (normalized to one in the figure). We take medians and quartiles over 50 trials. We use the Euler method to integrate the differential Equations.

Plotted are the median of the RMS error (shaded: intervals between first and third quartile) between the signal and the target in the first 30s after training. The error is computed using the normalized version of the signal, where the amplitude is set to 1. The sweep covers 80 equidistant values of $s_{\text{pert}}$ from 0 to 0.3 (displayed is the range up to 0.2).

For simulations that showed pathological spiking (more than 200 spikes per time step in the numerical simulation) we assigned an infinite error.

**Figure 3.10**

The parameters are as in Fig. 3.4a-c, lower panels, see “Figure details” in the main text.

**Figure 3.11**

Fig. 3.11a: The continuous networks obey Equation (3.11) (Equation (3.42)), they are endowed with the FORCE learning rule. The parameters of the network are $\lambda_x = 1 \frac{1}{s}$, $dt = 0.01s$, and the sparse matrix $A$ has a fraction $p = 0.1$ of nonzero entries, which are drawn from a Gaussian distribution with zero mean and variance.
3.6. Supplement: Learning universal computations with spikes

\[ \frac{g^2}{pN} \] with \( g = 1.51 \). The task from Fig. 3.3e serves as target signal. The learning rate is \( \alpha = 1 \), the learning time is \( T_t = 100.5s \). The network size \( N \) is swept from 10 to 100 in steps of 10. The figure shows the median of the RMS error (shaded: intervals between first and third quartile) between the signal and the target in the first 10s after training. The statistics are based on 100 trials per value of \( N \).

Fig. 3.11b,c: We use PCSNs with nonlinear dendrites, which encode continuous dynamics as generated by the rate networks in panel (a). \( \gamma_s \) is swept over 0.4, 0.25, 0.15, 0.1, 0.09, 0.075, 0.06, 0.05, 0.04, 0.025, 0.015, 0.01 and networks of size \( N \) are plotted for \( N = 70, 100, 200, 400 \). Fig. 3.11b shows the median of the RMS error (shaded: interval between first and third quartile) between the signal and the target in the first 10s after training. Fig. 3.11c shows the median of the mean spike rate per neuron (shaded: intervals between first and third quartile). The statistics are based on 20 trials per parameter combination.

Fig. 3.11d: The plot shows example patterns generated by continuous networks as used in panel (a) (blue trace) and PCSNs as used in panels (b,c) (red trace). For the continuous network \( N = 50 \), for the PCSNs \( \gamma_s = 0.01, N = 70, J = 50 \). The PCSN has a mean spike rate of 441 Hz.

**Figure 3.12**

Fig. 3.12a: The panel shows the last 100s of the PCSN simulation displayed in Fig. 3.11d. Total duration of the recall phase is 10000s, Fig. 3.11d shows the first 100s.

Fig. 3.12b: We use the same network parameters as in Fig. 3.11d and Fig. 3.12a, displayed are 10 learning trials with different randomly chosen initial connectivity. We compute a phase-shift corrected version of the RMS error in a sliding window of size 100s, for different starting points \( \tau \) of the sliding window. \( \tau \) is in the range from 1s to 10000s with step size 1s. The phase-shift corrected version of the RMS error is computed by

\[
\sqrt{\frac{1}{100s} \int_\tau^{\tau+100s} (signal(\tilde{t}) - target(\tilde{t} + \Delta))^2 d\tilde{t}}
\]

with the phase-shift \( \Delta \) of the target signal chosen such that the integral is minimal.

Fig. 3.12c: The parameters are the same as in Fig. 3.3h.
Chapter 4

Action selection in growing state spaces: Control of Network Structure Growth

4.1 Abstract

The dynamical processes taking place on a network depend on its topology. Influencing the growth process of a network therefore has important implications on such dynamical processes. We formulate the problem of influencing the growth of a network as a stochastic optimal control problem in which a structural cost function penalizes undesired topologies. We approximate this control problem with a restricted class of control problems that can be solved using probabilistic inference methods. To deal with the increasing problem dimensionality, we introduce an adaptive importance sampling method for approximating the optimal control. We illustrate this methodology in the context of formation of information cascades, considering the task of influencing the structure of a growing conversation thread, as in Internet forums. Using a realistic model of growing trees, we show that our approach can yield conversation threads with better structural properties than the ones observed without control.¹


4.2 Introduction

Many complex systems can be described as dynamic processes which are characterized by the topology of an underlying network. Examples of such systems are human interaction networks, where the links may represent transmitting opinions [OSFM07, DM11, CB15], habits [Cen10, FDGR+14], money [GK10, ACM16, GS16] or viruses [PSV01, EK02]. Being able to control, or just influence in some way, the dynamics of such complex networks may lead to important progress, for example, avoiding financial crises, preventing epidemic outbreaks or maximizing information spread in marketing campaigns.

The control of the dynamics on networks is a very challenging problem that has attracted significant interest recently [LSB11, CKM13, GLDB14, YTB+15]. Existing approaches typically consider network controllability as the controllability of the dynamical system induced by the underlying network structure. While it is agreed that network controllability critically depends on the network structure, the problem of how to control the network structure itself while it is evolving remains open.

The network structure is determined by the dynamics of addition and deletion of nodes and links over time. In this paper, we address the problem of influencing this dynamics in the framework of stochastic optimal control. The standard way to address these problems is through the Bellman equation and dynamic programming. Dynamic programming is only feasible in small problems and requires approximations when the state and action spaces are large. In the setting of network growth, this problem is more severe, since the state space increases (super-)exponentially with the number of nodes.

In order to deal with this curse of dimensionality, we propose to approximate the network growth control problem by a special class of stochastic optimal control problems, known as Kullback-Leibler (KL) control or Linearly-Solvable Markov Decision Problems (LMDPs) [KGO12, Tod09]. For this class of problems, one can use efficient adaptive importance sampling methods that scale well in high dimensions. The optimal solution for the KL-control problem tends to be sparse, so that only a few next states become relevant, effectively reducing the branching factor of the original problem. The obtained solution of the KL-control problem is then used to compute the optimal action in the original problem that does not belong to the KL-control class.

In the next section we present our proposed general methodology. We then apply it to a realistic problem: influencing the growth process of cascades in online forums, in order to maximize structural network measures that are connected to the quality of an online conversation thread. We conclude the paper with a discussion.
4.3 Optimal Network Growth as a Control Problem

We now formulate the network growth control problem as a stochastic optimal control problem. Let $x_t \in \mathcal{X}$, with $\mathcal{X}$ being the set of all possible network structures, denote the growing structure (state) of the network at time $t$ and let $P(x'|x,u)$ describe the network dynamics, where the control variable $u \in \mathcal{U}$ denotes possible actions we can perform in order to manipulate the network. Let us label the default action, which means not interacting with the system, with $u = 0$. We denote the corresponding dynamics without control as the uncontrolled process $p(x'|x) := P(x'|x,u = 0)$.

At each time-step $t$, we incur an arbitrary cost function on the network state $r(x,t)$ which is assigned when the state is reached. The state cost $r(x,t)$ penalizes network structures that are not convenient in the particular context under consideration. For example, if one wants to favour networks with large average clustering coefficient $C(x)$, then $r(x,t) = -C(x)$. Alternatively, one can consider more complex functions, such as the structural virality or Wiener index $\lbrack\text{MP88}\rbrack$, as proposed recently $\lbrack\text{GAHW15}\rbrack$, to maximize the influence in a social network. In general, any measure that can be (efficiently) computed from $x$ fits the presented framework.

Our objective is to find the control function $u(x,t) : \mathcal{X} \times \mathbb{N} \to \mathcal{U}$ which minimizes the total cost over a time horizon $T$, starting at state $x$ at initial time $t = 0$

$$J(x,t) \equiv \min_{u(\cdot)} \mathbb{E} \left( r(x,0) + \sum_{t' = t+1}^{T} r(x_{t'},t') \right)_{P(x_{t+1:T}|x,u(\cdot),t=0)},$$

(4.1)

where the expectation is taken with respect to the probability $P(x_{1:T}|x,u(\cdot),t = 0)$ over paths $x_{1:T}$ in the state space, given state $x$ at time $t = 0$ using the control-function $u(\cdot)$. The probability of a path is given by $P(x_{t+1:T}|x,u(\cdot),t = 0) = \prod_{s=t}^{T-1} P(x_{s+1}|x_s,u(x_s,s),s)$.

Computing the optimal control can be done by dynamic programming $\lbrack\text{Ber95}\rbrack$. We introduce the optimal cost-to-go

$$J(x,t) = \min_{u(\cdot)} \mathbb{E} \left( r(x,t) + J(x',t+1) \right)_{P(x'|x,u(\cdot),t)}. \quad (4.2)$$

which is an expectation of the cumulative cost starting at state $x$ and time $t$ and acting optimally thereafter. This can be computed using the Bellman equation

$$J(x,t) = \min_{u} \left( r(x,t) + \sum_{t'=t+1}^{T} r(x_{t'},t') \right)_{P(x_{t+1:T}|x,u(\cdot),t)}.$$  (4.3)

From $J(x,t)$, the optimal control is obtained by a greedy local optimization:

$$u^*(x,t) = \arg\min_{u} \left( r(x,t) + J(x',t+1) \right)_{P(x'|x,u(\cdot),t)}.$$  (4.4)
In general, the solution to Equation (4.3) can be computed recursively using dynamic programming [Ber95] for all possible states. This is however infeasible for controlling network growth, as the computation is of polynomial order in the number of states and the state space of networks increases super-exponentially on the number of nodes. E.g. for directed unweighed networks, there are $2^{N^2}$ possible networks with $N$ labelled nodes.

4.4 Approximating the network growth problem by a Kullback-Leibler control problem

In this section we present our main approach, which first computes the optimal cost-to-go on a relaxed problem and then uses it as a proxy for the original optimal cost-to-go. In the next subsection, we introduce the class of KL-control problems that we use as a relaxation. We then illustrate KL-control using a tractable example of tree growth. In Subsection 4.4.3, we explain how we can approximate the KL-control solution using the cross-entropy method. Finally, in Subsection 4.4.4 we show how we can use that result to compute the action selection in the original problem.

4.4.1 Kullback-Leibler control

In order to efficiently compute the optimal cost-to-go, we make the assumption that our controls directly specify the transition probabilities between two subsequent network structures, e.g. $P(x'|x, u(t)) \approx u(x'|x, t)$. Further, we define the natural growth process of the network (the uncontrolled dynamics) as a Markov chain with transition probabilities $p(x'|x)$. Because our influence on the network dynamics is limited, we add a regularization term to the total cost defined in Equation (4.1) that penalizes deviations from $p(x'|x)$. The approximated control cost becomes

$$
\mathcal{C}^\lambda_{KL} (x, t, u(\cdot)) = \lambda KL[u(x_{t+1:T}|x, t) \parallel p(x_{t+1:T}|x, t)] + r(x, t) + \left( \sum_{t'=t+1}^T r(x_{t'}, t') \right) u(x_{t+1:T}|x, t),
$$

with the KL-divergence

$$
KL[u(x_{t+1:T}|x, t) \parallel p(x_{t+1:T}|x, t)] = \log \left( \frac{u(x_{t+1:T}|x, t)}{p(x_{t+1:T}|x, t)} \right) \big|_{u(x_{t+1:T}|x, t)},
$$

which measures the closeness of the two path distributions, $p(x_{t+1:T}|x, t)$ and $u(x_{t+1:T}|x, t)$. The parameter $\lambda$ thereby regulates the strength of this penalization.
4.4. Approximating the network growth problem by a Kullback-Leibler control problem

With this assumption, the control problem consisting in minimizing $C^\lambda_{\text{KL}}$ w.r.t. the control $u(x'|x, t)$ belongs to the KL-control class and has a closed form solution [Tod09, KGO12]. The probability distribution of an optimal path $u^*_\text{KL}(x_{t+1:T}|x, t)$ that minimizes Equation (4.5) is

$$u^*_\text{KL}(x_{t+1:T}|x, t) = \frac{p(x_{t+1:T}|x, t)}{\langle \phi(x_{t+1:T}) \rangle_{p(x_{t+1:T}|x, t)}} \phi(x_{t+1:T}), \quad (4.6)$$

with

$$\phi(x_{t+1:T}) := \exp \left( -\lambda^{-1} \sum_{t'=t+1}^T r(x_{t'}, t') \right). \quad (4.7)$$

Plugging this into Equation (4.5) and minimizing gives the optimal cost-to-go

$$J^\lambda_{\text{KL}}(x, t) = r(x, t) - \lambda \log \langle \phi(x_{t+1:T}) \rangle_{p(x_{t+1:T}|x, t)}, \quad (4.8)$$

which can be numerically approximated using paths sampled from the uncontrolled dynamics $p(x_{t+1:T}|x, t)$.

The optimal control corresponding to Equation (4.4) corresponds to a state transition probability distribution that is obtained by marginalization in Equation (4.6). It is expressed in terms of the uncontrolled transition probability $p(x'|x)$ and the (exponentiated) optimal cost-to-go:

$$u^*_{\text{KL}}(x'|x, t) \propto \sum_{x_{t+2:T}} u^*_\text{KL}(x_{t+1} = x', x_{t+2:T}|x, t) = p(x'|x) \exp \left( -\frac{J^\lambda_{\text{KL}}(x', t+1)}{\lambda} \right). \quad (4.9)$$

This resembles a Boltzmann distribution with temperature $\lambda$ where the optimal cost-to-go takes the role of an energy. The effect of the temperature becomes clear: for high values of $\lambda$, $u^*_{\text{KL}}(x'|x, t)$ deviates only a little from the uncontrolled dynamics $p(x'|x)$, thus the optimal control has a weak influence on the system. In contrast, for low values of $\lambda$, the exponential in Equation (4.9) becomes very pronounced for the state(s) $x'$ with the smallest cost-to-go $J^\lambda_{\text{KL}}(x', t+1)$, suppressing the transition probabilities to suboptimal states $x'$. Thus the control has a very strong effect on the process. In the limit of $\lambda$ going to zero, the controlled process becomes deterministic, if $J^\lambda_{\text{KL}}(x', t+1)$ is not degenerate (meaning there is a unique state $x'$ which minimizes the optimal cost-to-go). In this case the control is so strong that it overpowers the noise completely.

We thus approximate our original (possibly difficult) control problem as a KL-control problem, parametrized by the temperature $\lambda$. The approximated optimal cost-to-go $J(x', t+1)$ of Equation (4.4) is replaced by the corresponding optimal cost-to-go of the KL-control problem $J^\lambda_{\text{KL}}(x', t+1)$ of Equation (4.9) and used to compute the action selection in the original problem.

4.4.2 A Tractable Example

We now present a tractable example amenable for exact optimal control computation. This example already belongs to the KL-control class, so no approximation is made. The purpose of this analysis is to show how different values of the temperature $\lambda$ may lead to qualitatively different optimal solutions and other interesting phenomena.

Let’s consider a tree that grows at discrete time-steps, starting with the root node at time $t = 0$. We represent the tree at time $t$ as a vector $x_t = (x_0, x_1, ..., x_t)$, where $x_t$ indicates the label of the parent of the node attached at time $t$. At every time-step, either the tree remains the same or a new node is attached to it. The root node has label 1 and the label 0 is specially used to indicate that no node was added at a given time-step (it is also the label of the parent of the root node). The nodes are labelled in increasing order as they arrive to the tree, so that at time-step $t$, for a tree with $k$ nodes, $k \leq t$, $x_t = 0, 1, \ldots, k$ corresponds to the parent of node $k + 1$ if a node is added or zero otherwise. Thus, the parent vector at time $t = 1$ is always $x_1 = (0, 1)$.

Our example is a finite horizon task of $T = 10$ time-steps and end-cost only. The end-cost implements two control objectives: it prefers trees of large Wiener index while penalising trees with many nodes (more than five, in this case). The Wiener index is the sum of the lengths of the shortest paths between all nodes in a graph. It is maximal for a chain and minimal for a star.

The uncontrolled process is biased to the root: new nodes choose to link the root with probability $3/5$ and uniformly otherwise. More precisely

$$
p(x_{t+1} = j|x_t) = \begin{cases} 
\frac{3}{5} & \text{for } j = 1 \\
\frac{2}{5||x_t||_0} & \text{for } (j = 0) \text{ or } j \text{ s.t. } (x_j > 1)
\end{cases}
$$

$$
r(x_t, t) = \begin{cases} 
-Wiener(x_t)\delta_{t,T} & \text{if } ||x_t||_0 < 5 \\
\delta_{t,T} & \text{otherwise}
\end{cases}
$$

where $||x||_0$ denotes the number of non-zero elements in $x$ and $Wiener$ the (normalized) Wiener index.

In this setting, the uncontrolled process $p$ tends to grow trees with more than five nodes with many of them attached to the root node, i.e. with low Wiener index. We want to influence this dynamics so that the target configuration, a chain of five nodes (maximal Wiener index) is more likely to be obtained.

Figure 4.1 (top) shows the state cost $r$ of the final tree that results from choosing the most probable control (MAP solution) as a function of the temperature $\lambda$. The exact solution is calculated using dynamic programming [KGO12]. We can differentiate three types of solutions, denoted as A, B and C in the figure.

For low temperatures (region A) the control aims to fulfill both control objectives: to find a small network with maximal Wiener index. The optimal strategy does not add nodes initially and then builds a tree of maximal Wiener index (see inset of
4.4. Approximating the network growth problem by a Kullback-Leibler control problem

Figure 4.1: Example of optimal control of tree growth. (Top): the state cost of the most probable solutions as a function of the temperature $\lambda$. In region A, the optimal strategy waits until the last time-steps and then grows a tree with maximal Wiener index. In region B, it builds a star of five nodes. Finally, in region C, it follows the uncontrolled dynamics and builds a star of ten nodes. (Left): for each region, the optimal probabilities $u^*(x_{t+1}|x_t)$ at $t = 1$ for the two actions which are initially available: no node addition (0) and adding a node to the root (1). In regions A, B the optimal control favours not adding a new node initially. The sequences on the right show how the tree grows. When a new node is added to the tree, it is coloured in red (color online).

initial controls in left column of the figure). This type of control (to wait while the target is far in the future) is reminiscent of the delayed choice mechanism described previously [Kap05]. This initial waiting period makes sense because if the chain of length 5 would be grown immediately, then at time 6 the size of 5 is already reached. If now an additional node attaches, then the final cost would be zero. However if one first waits and then grows the chain, an accidental node insertion before time 6 would not be so disastrous (actually it may help), as one can then just wait until time 7 to start growing the rest of the tree. So delaying the decision when to start growing the tree helps compensating accidental events.

For intermediate temperatures (region B), the initial control becomes less extreme, as we observe if we compare the left plots between regions A and B. For $\lambda \approx 0.07$, the solution that builds the tree with maximal Wiener index is no longer optimal, since it deviates too much from the uncontrolled dynamics. In region
B, the control aims to build a network of five nodes or less, but no longer aims to maximize the Wiener index. The control is characterized by an initial waiting period and the subsequent growth of a tree of five nodes, which are in this case all attached to the root node.

Finally, for high temperatures (region C, $\lambda > 0.4$), the control essentially ignores the cost $r$ and the optimal strategy is to add one node to the root at every time-step, following the uncontrolled process.

From these results we conclude that KL-control as a mechanism for controlling network growth can capture complex phenomena such as transitions between qualitatively different optimal solutions and delayed choice effects.

### 4.4.3 Sampling from the KL-optimally controlled dynamics

In this subsection, we explain how we can sample from the optimally controlled dynamics and thereby obtain an estimate of the optimal cost-to-go $J_{\text{KL}}^\lambda(x,t)$ of Equation (4.8).

The probability of an optimally controlled path, Equation (4.6), corresponds to the product of the uncontrolled dynamics by the exponentiated state costs. Hence a naive way to obtain samples from the optimal dynamics, would consist in sampling paths from the uncontrolled dynamics $p(x'|x)$ and weight them by their exponentiated state costs. Using these samples we can then compute expectations from the optimally controlled dynamics. We use that for any function $f(x_{t+1:T})$ we have:

$$\langle f(x_{t+1:T}) \rangle_{u_{\text{KL}}}(x_{s+1:T}|x,t) = \frac{\langle f(x_{t+1:T}) \phi(x_{t+1:T}) \rangle_{p(x_{t+1:T}|x,t)}}{\langle \phi(x_{t+1:T}) \rangle_{p(x_{t+1:T}|x,t)}} .$$

More precisely, provided a learned model or a simulator of the uncontrolled dynamics $p(x'|x)$, we generate $M$ sample paths $x^{(i)}_{t+1:T}$, $i = 1, \ldots, M$ from $p(x'|x)$ and compute the weights $\frac{\phi(x^{(i)}_{t+1:T})}{\hat{\phi}}$. The denominator thereby gives with Equation (4.8) an estimate of the optimal cost-to-go as

$$\langle \phi(x_{t+1:T}) \rangle_{p(x_{s+1:T}|x,t)} \approx \hat{\phi} := \frac{1}{M} \sum_{i=1}^{M} \phi(x^{(i)}_{t+1:T}) .$$

This method can be combined with resampling techniques [DC05,HSG06] to obtain unweighted samples $x^{\text{opt},(i)}_{t+1:T}$ from the optimal dynamics (for the numerical methods in this article, we have used structural resampling [DC05,HSG06]).

Using such a naive sampling method, however, can be inefficient, specially for low temperatures. While for high temperatures $\lambda$ basically all weights $\frac{\phi(x^{(i)}_{t+1:T})}{\hat{\phi}}$ are more or less equal, for low temperatures only a few samples with very large weights contribute to the approximation, resulting in very poor estimates.
4.4. Approximating the network growth problem by a Kullback-Leibler control problem

This is a standard problem in Monte Carlo sampling and can be addressed using the Cross-Entropy (CE) method [DBKMR05, KR16], which is an adaptive importance sampling algorithm that incrementally updates a baseline sampling policy or sequence of controls. Here we propose to use the CE method in the discrete formulation and use a parametrized Markov process \( \tilde{u}_\omega(x'|x, t) \), with parameters \( \omega \), to approximate \( u_{KL}^* \). The CE method in our setting alternates the following steps:

1. In the first step, the optimal control is estimated using \( M \) sample paths drawn from a parametrized proposal distribution \( \tilde{u}_\omega(x'|x, t) \).

2. In the second step, the parameters \( \omega \) are updated so that the proposal distribution becomes closer to the optimal probability distribution.

As a proposal distribution \( \tilde{u}_\omega(x'|x, t) \), we use

\[
\tilde{u}_\omega(x'|x, t) \propto p(x'|x) \exp \left( -\frac{\tilde{J}_{KL}(x', \omega(t))}{\lambda} \right),
\]  

which has the same functional form as the optimally controlled transition probabilities in Equation (4.9). The KL-optimal cost-to-go is thereby approximated by a linear sum of time-dependent feature vectors \( \psi_t^k(x) \)

\[
\tilde{J}_{KL}(x, \omega(t)) = \sum_k \omega_k(t)\psi_t^k(x).
\]  

The probability distribution of a path obtained using the optimal control, Equation (4.6), under this parametrization is given by

\[
\tilde{u}_\omega^*(x_{t+1:T}^{(i)} | x, t) \propto \tilde{u}_\omega(x_{t+1:T}^{(i)} | x, t) \frac{p(x_{t+1:T}^{(i)} | x, t)}{\tilde{u}_\omega(x_{t+1:T}^{(i)} | x, t)} \exp \left( -\lambda^{-1} \sum_{t'=t+1}^{T} r(x_{t'}^{(i)}, t') \right).
\]  

This shows that we can draw samples from the proposal distribution and reweight them with the combined weights

\[
w^{(i)} = \frac{p(x_{t+1:T}^{(i)} | x, t)}{\tilde{u}_\omega(x_{t+1:T}^{(i)} | x, t)} \phi(x_{t+1:T}^{(i)}).
\]  

The parameters \( \omega_k(t) \) of the importance sampler are initialized with zeros, which makes the initial proposal distribution equivalent to the uncontrolled dynamics. The procedure requires the gradients of \( \tilde{u}_\omega(x'|x, t) \) at each iteration. We describe the details of the CE method in 4.7.1.

We measure the efficiency of an obtained proposal control using the effective sample size (EffSS), which estimates how many effective samples can be drawn
from the optimal distribution. Given M samples with weights \( w^{(i)} \), the EffSS is given by
\[
\text{EffSS} = \frac{1}{M} \sum_{i=1}^{M} \left( w^{(i)} \right)
\]
\[
\left( \frac{1}{M} \sum_{i=1}^{M} w^{(i)} \right)^2.
\] (4.15)

If the weights \( w^{(i)} \) are all about the same value, the EffSS is high, indicating that many samples contribute to statistical estimates using the weighted samples. If all weights are equal, the EffSS is equal to the number of samples M. Conversely if the weights \( w^{(i)} \) have a large spread, the EffSS is low, indicating that only few independent samples contribute to statistical estimates. In the extreme case, when one weight is much larger then all others, the EffSS approaches 1.

### 4.4.4 Action selection using the KL-approximation

Once we have an estimate of the cost-to-go \( J_{KL}^{\lambda} \), we need to select an action \( u \in \mathcal{U} \) in the original control problem, which is not of the KL-control type. We select the optimal action according to
\[
u^*(x, t) \approx \arg\min_u \left( r(x, t) + J_{KL}^{\lambda}(x', t + 1) \right)_{P(x'|x, u, t)},
\] (4.16)
which requires the computation of \( J_{KL}^{\lambda}(x_{t+1}, t + 1) \) for every reachable state \( x_{t+1} \). In growing networks, the number of possible next states (the branching factor) increases quickly, and visiting all of them soon becomes infeasible.

In this subsection we highlight an important benefit of using the KL-approximation as a relaxation of the original problem: the optimally controlled process tends to discard many irrelevant states, specially for small values of \( \lambda \). This means that \( u_{KL}^*(x'|x, t) \) is sparse on \( x' \) (only a few next states are relevant for the task), since the cost \( J_{KL}^{\lambda}(x', t) \) is very large for the corresponding \( x' \) where \( u_{KL}^*(x'|x, t) \approx 0 \).

Let \( x_{t+1:2}^{\text{opt}} \) denote a trajectory sampled from the optimally controlled process, as described in the previous section. We compute \( u_{KL}^*(x'|x, t) \) using:
\[
\hat{u}_{KL}^*(x'|x, t) = \left( \delta_{x_{t+1}^{\text{opt}}, x'} \right) u_{KL}^*(x_{t+1:2}^{\text{opt}}|x, t),
\] (4.17)
where \( x_{t+1}^{\text{opt}} \) is the first element of the trajectory and \( \delta_{x_{t+1}^{\text{opt}}, x'} \) is the Kronecker delta which is equal one if \( x_{t+1}^{\text{opt}} \) is equal to \( x' \), and zero otherwise.

We then compute the optimal cost using Equation (4.9):
\[
J_{KL}^{\lambda}(x', t + 1) \sim -\log \left( \frac{\hat{u}_{KL}^*(x'|x, t)}{p(x'|x, t)} \right),
\] (4.18)
where we dropped a term which does not depend on \( x' \) and therefore plays no role in the minimization of Equation (4.16). The KL-approximation can help reducing
4.5 Application to Conversation Threads

We have described a framework for controlling growing graphs. We now illustrate this framework in the context of growing information cascades. In particular, we focus on the task of controlling the growth of online conversation threads. These are information cascades that occur, for example, in online forums such as weblogs [LMF+07], news aggregators [GKL08] or the synthesis of articles of Wikipedia [LTVK11]. In conversation threads, after an initial post appears, different users react writing comments either to the original post or to comments from other users, forming a cascade of messages. The figure shows an example of conversation thread taken from Slashdot about Google's AlphaGo. The control task is to influence the structure of the conversation thread (shown as a growing tree in the top-right) (color online).

As mentioned earlier, \( u_{KL}^*(x', x, t) \) tends to be more sparse for small values of \( \lambda \), when the KL-control problem is less noisy. In 4.7.2 we provide analytical details of the two extreme conditions, when \( \lambda \) is zero or infinite, respectively.

Figure 4.3: Our proposed control mechanism: in addition to the the threaded conversation, we highlight a comment (red node in the growing tree), suggested to be replied by the user. The choice of suggested comment, shown at the bottom of the page, is calculated using the method described in Section 4.4.3 (color online).

Figure 4.2 shows an example of a conversation thread, taken from Slashdot (www.slashdot.org). Users see a conversation thread using a similar hierarchical interface.

The task we consider is to optimize the structure of the generated conversation thread while it grows. The state is thus defined as a growing tree. We assume an underlying (not observed) population of users that keep adding nodes to this tree. Since we can not control directly what is the node that will receive the next comment, we propose the user interface as a control mechanism to influence indirectly the growth process. This can be done in different ways, for example, manipulating the layout of the comments. In our case, the control signal will be to recommend a comment (by highlighting it) to which the next user can reply. Figure 4.3 illustrates such a mechanism. The action selection strategy introduced in Section 4.4.4 is used to select the comment to highlight. Our goal is thus to modify the structure of a cascade in certain way while it evolves, by influencing its growth indirectly. It is known that the structure of online threads is strongly related with the complexity of the underlying conversation [GKL08, GBKB10].

To fully define our control problem, we need to specify the structural cost func-
4.5. Application to Conversation Threads

tion, the uncontrolled dynamics, i.e. the equivalent of equations (4.10) and (4.11) for this task, and a model of how an action (highlighting a node) changes the dynamics. Globally, this application differs from the toy example of Subsection 4.4.2 in some important ways:

1. The state-space is larger (threads typically receive more than 10 comments).
2. We choose as state-cost function the Hirsch index (h-index), which makes the control task highly non-trivial.
3. The original problem is not a KL-control problem. We use the action selection method described in Section 4.4.4 to control the growth of the conversation thread.

4.5.1 Structural Cost Function

We propose to optimize the Hirsch index (h-index) as structural measure. In our context, a cascade with h-index \( h \) has \( h \) comments each of which have received at least \( h \) replies. It is a sensible quantity to optimize, since it measures how distributed the comments of users on previous comments are. A high h-index prevents two extreme cases that occur in a rather poor conversation: the case where a small number of posts attract most of the replies, thus there is no interaction, and the case with deep chains, characteristic of a flame war of little interest for the community. Both cases have a low h-index, while a high h-index spreads the conversation over multiple levels of the cascade.

The h-index is a function of the degree sequence of all nodes in the tree, where the degree of a node is this case is the number of replies plus one, as there is also a link to the parent (replied comment or post). Therefore we use the degree histogram as features \( \psi_k(x) \) for the parametrized form of the optimal cost-to-go, Equation (4.13). That is, feature \( \psi_k(x) \) is the number of nodes with degree \( k \) in the tree \( x \) at time-step \( t \). We model the problem as a finite horizon task with end-cost. Thus, the state cost is defined as \( r(x, t) = -\delta_{t,T} \cdot h(x) \), where \( h(x) \) is the h-index of the tree \( x \).

4.5.2 Uncontrolled Dynamics for Online Conversation Threads

As uncontrolled dynamics, we use a realistic model that determines the probability of a comment to attract the replies of other users at any time, by means of an interplay between the following features:

- **Popularity** \( \alpha \): number of replies that a comment has already received.
- **Novelty** \( \tau \): the elapsed time since the comment appeared in the thread.
- **Root node bias** \( \beta \): characterizes the level of trendiness of the main post.

Such a model has proven to be successful in capturing the structural properties and the temporal evolution of discussion threads present in very diverse platforms [GKLK13]. Notice that these features \( \theta = (\alpha, \tau, \beta) \) should not be confused with the features \( \psi_t^k(x) \) used to encode the cost-to-go.

We represent the conversation thread as a vector of parents \( x_t = (x_0, x_1, \ldots, x_t) \). Given the current state of the thread \( x_t \), the uncontrolled dynamics attaches a new node \( t+1 \) to an existing node \( j \) with probability

\[
p_\theta(x_{t+1} = j|x_t) = \frac{1}{Z_{t+1}} \left( \text{deg}_{j,t} \alpha + \delta_{j,1} \beta + \tau^{t+1-j} \right)
\]

with \( Z_{t+1} \) a normalization constant, \( \text{deg}_{j,t} \) the degree of node \( j \) at time \( t \) and \( \delta_{j,1} \) the Kronecker delta function, so parameter \( \beta \) is only nonzero for the root.

Given a dataset composed of \( S \) threads \( D := \{x^{(1)}, \ldots, x^{(S)}\} \) with respective sizes \( |x^{(k)}|, k \in \{1, \ldots, S\} \), the parameter vector \( \theta \) can be learned by minimizing

\[
-\log \mathcal{L}(D; \theta) = -\sum_{k=1}^{S} \sum_{t=2}^{|x^{(k)}|} \log p_\theta(x^{(k)}_{t+1}|x^{(k)}_t).
\]

We learn the parameters using the Slashdot dataset, which consists of \( S = 9,820 \) threads, containing more than \( 2 \cdot 10^6 \) comments among 93,638 users. In Slashdot, the most relevant feature is the preferential attachment, as detailed in [GKLK13]. This will have implications in the optimal control solution, as we show later.

4.5.3 Control interaction

The control interaction is done by highlighting a single comment of the conversation. We assume a behavioural model for the user inspired by [CZTR08], where the user looks at the highlighted comment and decides to reply or not. For simplicity, we assume that the user chooses the highlighted comment with a fixed probability \( p' = \alpha/(1+\alpha) \) and with probability \( 1-p' \) she chooses to ignore it. If the highlighting of the comment is ignored, the thread grows according to the uncontrolled process. Therefore, \( \alpha \) parametrizes the strength of the influence the controller has on the user. For \( \alpha \to \infty \), we can fully control the behaviour and for \( \alpha = 0 \), the thread evolves according to the uncontrolled process. A typical control would have a small \( \alpha \) as usually the influence of an controlling agent on a social systems is weak.

4.5.4 Experimental Setup

To evaluate the proposed framework we use a simulated environment, without real users. We consider a finite horizon task with \( T = 50 \) with the goal to maximize the h-index at end-time, starting from a thread with a single node as initial condition.
4.5. Application to Conversation Threads

The state-space consists of $50! \approx 3^{64}$ states. The thread grows in discrete time-steps. At each time-step, a new node is added to the thread by a (simulated) user. For that, we first choose which node to highlight (optimal action) as described in Section 4.4.4 using Equation (4.16). We then simulate the user as described in section 4.5.3, so the highlighted node is selected with probability $p' = \alpha / (1 + \alpha)$ as the parent of the new node. Otherwise, with probability $1 - p'$, the user ignores the highlighted node and the parent of the new node is chosen according to the Slashdot model, Equation (4.19). This type of model predictive control is repeated until the end time.

4.5.5 Experimental Results

We first analyse the performance of the adaptive importance sampling algorithm described in Section 4.4.3 for different fixed values of $\lambda$.

Figure 4.4 shows the effective sample size (EffSS), Equation (4.15) as a function of the number of iterations of the CE method. We observe that the EffSS increases to reach a stable value. As expected, large temperature (easier) problems result in higher values of EffSS. We can also see that, even for hard problems with low temperature, the obtained EffSS is significantly larger than zero, which allows us to compute the KL-optimal control. In general, the curves are less smooth for smaller values of $\lambda$, because a few qualitatively better samples dominate the EffSS, resulting in higher variance. On the other hand we also observe that the EffSS never reaches...

**Figure 4.5:** The learned importance sampler: The figure shows the time-dependent parameters of the learned expected cost-to-go for $\lambda = 0.2$. Each pixel is the parameter of a feature at a certain time. The features are the degrees of the parent node after the new child attaches. The colour represents the weight of the parameter. Large negative weights (pixels in blue colour) stand for a low cost and thus a desirable state, while large positive weights (red pixels) stand for high cost and thus undesirable states. At all times there is a desirable degree which the parent should have and higher as well as lower degrees are inhibited. This desirable degree is small at early times and becomes larger at later times (color online).

100%. This is expected, as this would mean that our parametrized importance sampler perfectly resembles the optimal control, and this is not possible due to the approximation error introduced by the use of features.

We can better understand the learned control by analysing the linear coefficients of the parametrized optimal cost-to-go, Equation (4.13), for this problem. Figure 4.5 shows the feature weights $\omega_k(t)$, at different times $t = 1, \ldots, T$, after convergence of the CE method. Feature $k$ corresponds to the number of nodes with degree $k$ in the tree, after a new node arrives. The parent node to which the new node has attached is thereby the only node whose degree changes (the degree increases by 1). Thus a high weight for a feature which measures the number of nodes with a certain degree $k$ results in a low probability of attaching to a node with degree $k - 1$. Conversely low, or large negative weights thus correspond to nodes which have a high probability of becoming the parent of the next node which is added. We observe that there is an intermediate preferred degree (large negative weight, in blue). This is the preferred degree of the parent of the new node, and this preferred degree increases with time, reaching a value of 5 at $t = 50$.

Does this strategy make sense? The maximum h-index of a tree of 50 nodes is
7, and it is achieved if 6 nodes have exactly 7 children and one node has 8. However, achieving such a configuration requires a very precise control. For example, increasing too much the degree of a node, say up to 9, prevents the maximum h-index to be reached, as there are not enough links left, due to the finite horizon. Thus, in this setting, steering for the maximal possible h-index is not optimal. The controller prefers all parents to have a degree of 5 and not less, but also not much more. As having more than five parents with degree at least five will result in an h-index of 5 we conclude that the control seems to aim for a target h-index of 5, while preventing wasting links to higher or lower degree nodes, which would not contribute to achieve that target.

The interpretation of why the preferred degree increases with time involves the uncontrolled dynamics. Remember that the most relevant term in Equation (4.19) for the considered dataset corresponds to the preferential attachment, parametrized by \( \alpha \). This term boosts high-degree nodes to get more links. If this happens, most of the links end up attached to a few parents, and this effect can only be suppressed by a strong control. The controller prevents that self-amplifying effect by aiming initially for an overall low degree, preventing a high impact of the preferential attachment. This keeps the process controllable and allows for a more equal distribution of the links.

After having evaluated the sampling algorithm, we evaluate the proposed mechanism for actual control of the conversation thread. As described in Section 4.4.4, in our simulated scenario, we highlight the node as the parent which minimizes the computed expected cost-to-go.

Figure 4.6 shows the evolution of the h-index using different control mechanisms. The blue curve shows how the h-index changes under the uncontrolled dynamics. On average, it reaches a maximum of about 3.7 after 50 time steps. In green, we show the evolution of the h-index under a KL-optimal controlled case, for temperature \( \lambda = 0.2 \). As expected, we observe a faster increase, on average, than using the uncontrolled dynamics. The maximum is about 4.7.

The red and black curves show the evolution of the h-index using the control mechanism described in Subsections 4.4.4 and 4.5.3, where we select actions using the expected cost-to-go \( J_{KL}^\lambda \) of the KL-optimal control with \( \lambda = 0.2 \), for \( \alpha = 1 \) and \( \alpha = 0.5 \), respectively. In both cases the obtained h-index is even higher than the one obtained with the KL-control relaxation. Therefore, the objective for this task, to increase the h-index, can be achieved through our action selection strategy. As expected, a stronger interaction strength \( \alpha = 1 \) leads to higher h-indices than a lower strength \( \alpha = 0.5 \).

Finally, in Figure 4.7 we show examples of a real discussion thread from the dataset (Slashdot), a thread generated from the learned model (uncontrolled process) and one resulting from applying our action selection strategy. The latter has higher h-index.

**Figure 4.6:** Evaluation of the actual control: uncontrolled dynamics (blue), KL-optimally controlled dynamics (green) action selection based control for $\alpha = 1$ (red) and $\alpha = 0.5$ (black). The KL-optimally controlled dynamics, which optimize the sum of the $\lambda$-weighted KL-term and the end cost, shifts the final mean value from about 3.7 to about 4.7. The action selection based control, which is aiming to optimize the end cost only, is able to shift the h-index to even higher values than the KL-optimal control. For the controlled dynamics, $\lambda = 0.2$ for all three cases. To compute the control in each time-step we sample 1000 trajectories. The statistics where computed using 1000 samples for each of the three cases.

### 4.6 Discussion

We have addressed the problem of controlling the growth process of a network using stochastic optimal control with the objective to optimize a structural cost that depends on the topology of the growing network. The main difficulty of such a problem is the exploding size of the state space, which grows (super-)exponentially with the number of nodes in the network and renders exact dynamic programming infeasible.

We have shown that a convenient way to address this problem is using KL-control, where a regularizer is introduced which penalizes deviations from the natural network growth process. One advantage of this approach is that the optimal control can be solved by sampling. The difficulty of the sampling is controlled by the strength of the regularization, which is parametrized by a temperature parameter $\lambda$: for high temperatures the sampling is easy, while for low temperatures, it becomes hard. This is in contrast to standard dynamic programming, whose complexity is directly determined by the number of states and independent of $\lambda$.

In order to tackle the more challenging low temperature case, we have in-
4.6. Discussion

Figure 4.7: Examples of threads. A thread from the data (Slashdot), an uncontrolled thread generated from the model and a controlled thread. The nodes that contribute to the h-index are coloured in yellow. The h-index for the data and the uncontrolled thread is 4 and 6 for the controlled one (color online).

introduced a feature-based parametrized importance sampler and used adaptive importance sampling for optimizing its parameters. This allows us to sample efficiently in the low temperature regime. For control problems which cannot directly be formulated as KL-control problems, we have proposed to use the solution of a related KL-control problem as a proxy to estimate the effective values of possible next network states. These expected effective values are subsequently used in a greedy strategy for action selection in the original control problem. This action selection mechanism benefits from the sparsity induced by the optimal KL-control solution.

We have illustrated the effectiveness of our method on the task of influencing the growth of conversation cascades. Our control seeks to optimize the structure of the cascade, as it evolves in time, to maximize the h-index at a final time. This task is non-trivial and characterized by a sparse, delayed reward, since the h-index remains constant during most of the time, and therefore a greedy strategy is not possible.

Our approach for controlling network growth is inspired in recent approaches to optimal decision-making with information-processing constraints [Tod09, TP11, KGO12, TT12, RTV12]. The Cross-Entropy method has been explored previously in the continuous case [KR16]. The continuous formulation of this class of problems has been used in robotics, using parametrized policies [TBS10, LK13, GKPN14]. In economics, the question of altering social network structure in order to optimize utility has been addressed mainly from a game theoretical point of view, under the
name of strategic network formation [JW02, BJ07]. To the best of our knowledge, the problem of network formation has not yet been addressed from a stochastic optimal control perspective.

The standard approach to address the problem of controlling a complex, networked system is to directly try to control the dynamics on the network [LSB11, CKM13, GLDB14, YTB+15]. This approach considers the classical notion of structural controllability as the capability of being driven from any initial state to any desired final state within finite time. Optimal control in thus referred to the situation where a network can be fully controlled using only one driving signal. This idea is also prevalent in the influence maximization problem in social networks [KKT03, FDGR+14, FWR+15], which consists in finding the subset of driver (most influential) nodes in a network.

Since the controllability of the dynamics on the network depends crucially on the topology, several works have considered the idea of changing the network structure in some way that favours structural controllability.

For example, the perturbation approach introduced in [WNLG12] looks for the minimum number of links that needs to be added so that the perturbed network can be fully controlled using a single input signal. In [HLSX15], a method to enhance structural controllability of a directed network by changing the direction of a small fraction of links is proposed. More recently, [WYS16] analyzed node augmentation of directed networks while insisting that the minimum number of drivers remains unchanged.

The main difference between our approach and these approaches is that, rather than considering the controllability of the dynamical system on the underlying network, our optimal control task is defined on the structure of the network itself, regardless of the dynamical system defined on it. In some sense, our results complement these approaches. For example, one could use our optimal control approach to shape the growth of the network in a way that the structural controllability, understood as the state cost function, is optimized.

4.7 Supplement: Action selection in growing state spaces

4.7.1 Adaptive Importance Sampling for KL-Optimal Control Computation using the Cross-Entropy method

Here we show how the time-dependent weights $\omega_k(t)$ of the importance sampler are updated such that $\tilde{u}_n(x'|x, t)$ becomes closer to the optimal sampling distribution. This corresponds to the second step of the Cross-Entropy method described in Subsection 4.4.3. For clarity in the derivations, we will replace $p(x_{1:T}|x, 0)$ and $u_{KL}^*(x_{1:T}|x, 0)$ by $p$ and $u_{KL}$, respectively, in the expectations. The closeness of the
4.7. Supplement: Action selection in growing state spaces

two distributions \(\tilde{u}_\omega(x'|x, t)\) and \(u^*_\text{KL}(x'|x, t)\) can be measured as the cross entropy between the path \(x_{1:T}\) probabilities under these two Markov processes:

\[
\text{KL}[u^*_\text{KL}(x_{1:T}|x, 0) \| \tilde{u}_\omega(x_{1:T}|x, 0)] = \left\langle \log \frac{u^*_\text{KL}(x_{1:T}|x, 0)}{\tilde{u}_\omega(x_{1:T}|x, 0)} \right\rangle_{u^*_\text{KL}} \\
= -\left\langle \log \tilde{u}_\omega(x_{1:T}|x, 0) \right\rangle_{u^*_\text{KL}} + \text{const.} = -\text{D}(\omega),
\]

where the constant term \(\left\langle \log u^*_\text{KL}(x_{1:T}|x, 0) \right\rangle_{u^*_\text{KL}}\) is dropped.

We minimize Equation (4.20) by gradient descent. At iteration \(l\), the gradient \(D(\omega(t))\) with respect to \(\omega_k(t)\) is given by

\[
\frac{\partial D(\omega(t))}{\partial \omega_k(t)} = -\left\langle \frac{\partial}{\partial \omega_k(t)} \log \tilde{u}_\omega(0)(x_{1:T}|x, 0) \right\rangle_{u^*_\text{KL}}
\]

where

\[
\tilde{u}_\omega(0)(x_{1:T}|x, 0) = \frac{1}{Z} p(x_{1:T}|x, 0) \prod_{t=0}^{T-1} \exp \left( -\frac{\tilde{J}_\text{KL}(x_{t+1}, \omega(t))}{\lambda} \right)
\]

\[
Z = \left\langle \prod_{t'=0}^{T-1} \exp \left( -\frac{\tilde{J}_\text{KL}(x_{t'+1}, \omega(t'))}{\lambda} \right) \right\rangle_p
\]

with the normalization constant \(Z\). This leads to

\[
\frac{\partial D(\omega(t))}{\partial \omega_k(t)} = -\left\langle \frac{\partial}{\partial \omega_k(t)} \left( \log p(x_{1:T}|x, 0) - \sum_{t'=0}^{T-1} \frac{\tilde{J}_\text{KL}(x_{t'+1}, \omega(t'))}{\lambda} - \log Z \right) \right\rangle_{u^*_\text{KL}}
\]

(4.21)

where we can drop the first term as it is independent of \(\omega(t)\). The second term can be evaluated using the definition of \(\tilde{J}_\text{KL}\), Equation (4.13).

**Algorithm 1** Cross-Entropy Method for KL-control

**Require:** importance sampler $\tilde{u}_{\omega}$, 
feature space $\psi(\cdot)$, 
number of samples $M$, 
learning rate $\eta$

$l \leftarrow 0$ 
$\omega_k^{(l)}(t) \leftarrow 0$, Initialize weights for all $k, t, l$ 
$x_{t+1:T}^{(l)} \leftarrow$ draw $M$ sample trajectories $\sim \tilde{u}_{\omega^{(l)}}$, $i = 1, \ldots, M$

repeat
  
  compute gradient $\frac{\partial D(\omega^{(l)})}{\partial \omega_k(t)}$ using Equation (4.21)
  
  $\omega_k^{(l+1)}(t) \leftarrow \omega_k^{(l)}(t) + \eta \frac{\partial D(\omega^{(l)})}{\partial \omega_k(t)}$ for all $k, t, l$

  $x_{t+1:T}^{(l)} \leftarrow$ draw $M$ samples $\sim \tilde{u}_{\omega^{(l+1)}}$

  $l \leftarrow l + 1$

until convergence

Further, plugging in $Z$ we get

$$
\frac{\partial D(\omega^{(l)})}{\partial \omega_k(t)} = \lambda^{-1} \left\langle \psi_k^t(x_{t+1}) \right\rangle_{\omega_k^u} + \frac{\partial}{\partial \omega_k(t)} \left\langle \log \left( \prod_{t'=0}^{T-1} \exp \left( -\frac{\bar{J}_{\text{KL}}(x_{t'+1}, \omega(t'))}{\lambda} \right) \right) \right\rangle_{\omega_k^p}
$$

$$= \lambda^{-1} \left\langle \psi_k^t(x_{t+1}) \right\rangle_{\omega_k^u} + \frac{\partial}{\partial \omega_k(t)} \log \left( \prod_{t'=0}^{T-1} \exp \left( -\frac{\bar{J}_{\text{KL}}(x_{t'+1}, \omega(t'))}{\lambda} \right) \right)_{\omega_k^p}
$$

$$= \lambda^{-1} \left\langle \psi_k^t(x_{t+1}) \right\rangle_{\omega_k^u} + \frac{\partial}{\partial \omega_k(t)} \left( \frac{1}{Z} \left\langle \prod_{t'=0}^{T-1} \exp \left( -\frac{\bar{J}_{\text{KL}}(x_{t'+1}, \omega(t'))}{\lambda} \right) \right\rangle_{\omega_k^p} \right)
$$

$$= \lambda^{-1} \left( \left\langle \psi_k^t(x_{t+1}) \right\rangle_{\omega_k^u} - \frac{1}{Z} \left\langle \psi_k^t(x_{t+1}) \prod_{t'=0}^{T-1} \exp \left( -\frac{\bar{J}_{\text{KL}}(x_{t'+1}, \omega(t'))}{\lambda} \right) \right\rangle_{\omega_k^p} \right)
$$

$$= \lambda^{-1} \left( \left\langle \psi_k^t(x_{t+1}) \right\rangle_{\omega_k^u} - \left\langle \psi_k^t(x_{t+1}) \tilde{u}_{\omega^{(l)}}(x_{1:T}|x,0) \right\rangle \right)
$$

$$= \lambda^{-1} \left( \left\langle \psi_k^t(x_{t+1}) \right\rangle_{\omega_k^u} - \frac{p(x_{t+1}|x,0) \phi(x_{1:T}) \left\langle \psi_k^t(x_{t+1}) \right\rangle_{\omega_k^{(l)}} \tilde{u}_{\omega^{(l)}}(x_{1:T}|x,0) \right\rangle \right) - \left\langle \psi_k^t(x_{t+1}) \tilde{u}_{\omega^{(l)}}(x_{1:T}|x,0) \right\rangle,
$$

(4.22)

The update rule for the parameters becomes

$$\omega_k^{(l+1)}(t) = \omega_k^{(l)}(t) + \eta \frac{\partial D(\omega^{(l)})}{\partial \omega_k(t)},$$

(4.23)
4.7. Supplement: Action selection in growing state spaces

for some learning rate $\eta$. Algorithm 1 summarizes the CE method applied to this context.

4.7.2 Analyzing the KL-optimal cost-to-go based action selection

We have introduced an action selection framework which is based on an approximation of the optimal cost-to-go $J(x', t)$ by the optimal cost-to-go $J_{KL}^\lambda(x', t+1)$ of a parametrized family of KL-control problems which share the same state cost $r(x, t)$.

Why is this a good idea? Consider the two extreme cases where the temperature $\lambda$, which parametrizes the family of equivalent KL-control problems, is zero or infinite, respectively.

**Extreme case $\lambda \to 0$ (zero temperature):** The total cost in the KL-control problem becomes equal to the total cost in the original control problem, Equation (4.1), as the KL term vanishes. The KL-optimal control becomes deterministic:

$$\lim_{\lambda \to 0} u_{KL}^*(x'|x, t) = \lim_{\lambda \to 0} \frac{p(x'|x) \exp \left(- \frac{J_{KL}^\lambda(x', t+1)}{\lambda} \right)}{Z} = \begin{cases} 1 & \text{for } x' = \arg\min J_{KL}^\lambda(x', t+1), \\ 0 & \text{otherwise} \end{cases}$$

(4.24)

where $Z$ is a normalization constant.

Thus, for $\lambda \to 0$, the KL-control problem becomes identical to the original problem if the system is fully controllable, i.e. for every $t, x$ and $\tilde{x}$ there is a $u_{\tilde{x}} \in \mathcal{U}$ such that $p(x'|x, t, u_{\tilde{x}}) = \delta_{\tilde{x}, x'}$.

**Extreme case $\lambda \to \infty$ (infinite temperature):** For this case, using Equation (4.8) we get

$$J_{KL}^\infty(x, t) = \lim_{\lambda \to \infty} J_{KL}^\lambda(x, t)$$

$$= r(x, t) - \lim_{\lambda \to \infty} \lambda \log \left( \exp \left( -\lambda^{-1} \sum_{t'=t+1}^{T} r(x_{t'}, t') \right) \right)_{p(x_{t+1:T}|x, t)}$$

$$= r(x, t) + \left( \sum_{t'=t+1}^{T} r(x_{t'}, t') \right)_{p(x_{t+1:T}|x, t)}.$$

Using Equation (4.1) and the definition of the uncontrolled dynamics, we can write

$$J_{KL}^\infty(x, t) = r(x, t) + \left( \sum_{t'=t+1}^{T} r(x_{t'}, t') \right)_{p(x_{t+1:T}|x, 0, t)} = \mathcal{C}(x, t, 0).$$

(4.25)
Thus, for $\lambda \to \infty$, the KL-optimal cost-to-go becomes equal to the total cost in the original control problem under the uncontrolled dynamics (using $u = 0$). Having this Equation (4.16) can be written as

$$u^*(x, t) \approx \arg\min_u \left( r(x, t) + \left( c(x', t + 1, 0) \right) p(x' | x, u, t) \right). \tag{4.26}$$

In this case, the action selection is equivalent to optimize an expected total cost assuming the system will evolve according to the free dynamics in the future. Thus the infinite temperature control can be used if one wants to guarantee that the obtained solution will not be worse than the solution obtained with zero control. Choosing a lower $\lambda$, however, might in practice work better (as we also have shown in Section 4.5) but has no theoretical guarantee.

We can conclude that our action selection strategy is meaningful in the two extreme cases, $\lambda \to \infty$ and $\lambda \to 0$. Also this analysis suggests that, if the available set of actions $u \in U$ offers a strong control over the system dynamics, it is more convenient to use a $J^\lambda_{KL}$ with a low temperature $\lambda$. 
Chapter 5

Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.1 Abstract

For Path Integral control problems a representation of an optimally controlled dynamical system can be formally computed and serve as a guidepost to learn a parametrized policy. The recently proposed Path Integral Cross-Entropy (PICE) method tries to exploit this, but is hampered by poor sample efficiency. We identify PICE as the infinite smoothing limit of a newly developed smoothing technique that is applied to the cost function. The sample efficiency problems that PICE suffers disappear for finite levels of smoothing. We propose the model free algorithm ASPIC (Adaptive Smoothing of Path Integral Control) that utilizes smoothing to speedup convergence of policy optimization. For zero smoothing this method becomes a greedy optimization of the cost. Using theoretical and numerical studies we find that intermediate levels of smoothing are optimal, which renders the new method superior to both PICE and direct cost-optimization.
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.2 Introduction

How to choose an optimal action? For noisy dynamical systems stochastic optimal control theory provides a framework to answer this question. Optimal control is framed as an optimization problem to find the control that minimizes an expected cost function. For non-linear dynamical systems that are continuous in time and space this problem in general hard.

A method that has proven to work well is to introduce a parametrized policy like a neural network \([MKS^{+15},HSL^{+17},DCH^{+16}]\) and greedily optimize the expected cost using gradient descent \([Wil92,PS08,SLA^{+15},HSL^{+17}]\). To achieve a robust decrease of the expected cost it is important to ensure that in each step, the updated policy stays in the proximity of the old policy \([DCH^{+16}]\). This can be achieved by enforcing a trust region constraint \([PMA10,SLA^{+15}]\) or using adaptive regularization that punishes strong deviations of the new policy from the old policy \([HSL^{+17}]\). With this method one can achieve remarkable results \([HSL^{+17}]\). However these policy gradient methods can be very time-consuming, as in each iteration of the algorithm, samples from the controlled system have to be computed. This makes it difficult to apply these methods to systems that are costly to simulate, or where no model is available and the samples have to be generated by the real system. We want to increase the convergence rate of policy optimization in order to reduce the number of simulations needed to compute a control policy.

To this end we focus on the restricted class of Path Integral control problems that offers an alternative approach to direct cost optimization and explore if this allows to speed up policy optimization. This class of control problems allows for arbitrary non-linear dynamics and state cost but it requires a linear dependence of the control on the dynamics and a quadratic control cost \([Kap05,KR16,BK14,TK15]\). These restrictions allow to obtain an explicit expression for the probability-density of optimally controlled system trajectories. Through this, an information-theoretical measure of the deviation of the current control policy from the optimal control can be calculated \([KR16]\). The Path Integral Cross-Entropy (PICE) method \([KR16]\) proposes to use this measure as a pseudo objective for policy optimization. This might be a superior approach to greedy cost-optimization as it uses the optimal policy as a guidepost.

However, there is yet no comparative study on whether PICE actually offers an advantage over direct cost optimization; and, in its original form \([KR16]\), the PICE method does not scale well to more complex problems because the PICE gradient is hard to estimate if the current controller is not close enough to the optimal control \([RK17]\). Furthermore the PICE method has been introduced with standard gradient descent and does not use trust regions to ensure robust updates, which has been shown to be effective for policy optimization \([DCH^{+16}]\).

In this work we propose and study a new kind of smoothing technique for the cost function that allows to interpolate between the optimization of the direct cost
and the PICE objective. Optimizing this smoothed cost using a trust-region-based method yields an approach that is efficient and does not suffer from the feasibility issues of PICE. Our work is based on recently proposed smoothing techniques to speed up convergence in deep neural networks [COO+17]. We adapt this smoothing technique to Path Integral control problems. In contrast to [COO+17], smoothing of Path Integral control problems can be solved analytically and we obtain an expression of the gradient that can directly be computed from Monte Carlo samples. The strength of smoothing can be regulated by a parameter. In the limits of this smoothing parameter we recover the PICE method for infinitely strong smoothing and direct cost optimization for zero smoothing, respectively. As in [COO+17], the minimum of the smoothed cost, thus the optimal control policy, remains the same for all levels of smoothing.

We provide a theoretical argument why smoothing is expected to speed up optimization and conduct numerical experiments on different control tasks, which show that this accelerative effect exists indeed. For this we develop an algorithm called ASPIC (Adaptive Smoothing of Path Integral Control) that uses cost smoothing to speed up policy optimization. The algorithm adjusts the smoothing parameter in each step to keep the variance of the gradient estimator at a predefined level. To ensure robust updates of the policy, ASPIC enforces a trust region constraint; similar to [SLA+15] this is achieved with natural gradient updates and an adaptive stepsize. Like other policy gradient based methods [Wil92,PS08,SLA+15,HSL+17] ASPIC is model free.

Many policy optimization algorithms are updating the control policy based on a direct optimization of the cost; examples are the TRPO [SLA+15] or PIREPS [GKPN14], where the later is particularly developed for Path Integral control problems. The main novelty of this work is the application of the idea of smoothing as introduced in [COO+17] to Path Integral control problems. This allows to outperform direct cost-optimization and achieve faster convergence rates with only a negligible amount of computational overhead.

5.3 Path integral control problems

Consider the (multivariate) dynamical system

\[ \dot{x}_t = f(x_t, t) + g(x_t, t)(u(x_t, t) + \xi_t), \]  

(5.1)

with initial condition \( x_0 \). The control policy is implemented in the control function \( u(x, t) \), which is additive to the white noise \( \xi_t \) which has variance \( \frac{\nu}{\pi_t} \).

Given a control function \( u \) and a time horizon \( T \), this dynamical system induces a probability distribution \( p_u(\tau) \) over state trajectories \( \tau = \{x_t|\forall t : 0 < t \leq T\} \) with initial condition \( x_0 \).
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

We define the regularized expected cost

\[
C(p_u) = \langle V(\tau) \rangle_{p_u} + \gamma KL(p_u || p_0),
\]

with \( V(\tau) = \int_0^T V(x_t, t) dt \), where the strength of the regularization \( KL(p_u || p_0) \) is controlled by the parameter \( \gamma \).

The Kullback-Leibler divergence \( KL(p_u || p_0) \) puts high cost to controls \( u \) that bring the probability distribution \( p_u \) far away from the uncontrolled dynamics \( p_0 \) where \( u(x_t, t) = 0 \). We can also rewrite the regularizer \( KL(p_u || p_0) \) directly in terms of the control function \( u \) by using the Girsanov theorem (compare [TK15]):

\[
\log \frac{p_u(\tau)}{p_0(\tau)} = \frac{1}{\nu} \int_0^T \left( \frac{1}{2} u(x_t, t)^T u(x_t, t) + u(x_t, t)^T \xi_t \right) dt.
\]

The regularization then takes the form of a quadratic control cost

\[
KL(p_u || p_0) = \left\langle \frac{1}{\nu} \int_0^T \left( \frac{1}{2} u(x_t, t)^T u(x_t, t) + u(x_t, t)^T \xi_t \right) dt \right\rangle_{p_u}.
\]

where we used that \( \langle u(x_t, t)^T \xi_t \rangle_{p_u} = 0 \). This shows that the regularization \( KL(p_u || p_0) \) puts higher cost for large values of the controller \( u \).

The Path Integral control problem [Kap05, TK15, KR16] is to find the optimal control function \( u^* \), that minimizes the regularized cost \( C(p_u) \)

\[
u^* = \arg \min_u C(p_u).
\]

For a more complete introduction to Path Integral control problems, see [TK15, KR16].

### 5.3.1 Direct cost-optimization using gradient descent

A standard approach (see e.g. [HSL+17, Wil92, SLA+15]) to find an optimal control function is to introduce a parametrized controller \( u_\theta(x_t, t) \). This parametrizes the path probabilities \( p_{u_\theta} \) and allows to optimize the expected cost \( C(p_{u_\theta}) \) (5.2) using stochastic gradient descent on the cost function:

\[
\nabla_\theta C(p_{u_\theta}) = \left\langle S^\gamma_{p_{u_\theta}}(\tau) \nabla_\theta \log p_{u_\theta}(\tau) \right\rangle_{p_{u_\theta}},
\]

with the stochastic cost \( S^\gamma_{p_{u_\theta}}(\tau) := V(\tau) + \gamma \log \frac{p_{u_\theta}(\tau)}{p_0(\tau)} \) (see appendix 5.10 for details).
5.3.2 The PICE method

An alternative approach to direct cost-optimization was introduced as the PICE method in [KR16]. It uses that we can obtain an expression for $p_{u^*}$, the probability density of state trajectories induced by a system with the optimal controller $u^*$:

$$p_{u^*} = \arg \min_{p_u} C(p_u),$$

with $C(p_u)$ given by equation (5.2). Finding $p_{u^*}$ is an optimization problem over the space of all probability distributions $p_u$ that are induced by the controlled dynamical system (5.1). It has been shown [BK14, TK15] that we can solve this by replacing the minimization over $p_u$ with a minimization over all path probability distributions $p$:

$$p_{u^*} \equiv p^* = \arg \min_{p} C(p) = \arg \min_{p} \langle V(\tau) \rangle_{p} + \gamma KL(p||p_0)$$

$$= \frac{1}{Z} p_0(\tau) \exp \left( -\frac{1}{\gamma} V(\tau) \right). \quad (5.5)$$

with the normalization constant $Z = \langle \exp \left( -\frac{1}{\gamma} V(\tau) \right) \rangle_{p_0}$.

The PICE algorithm [KR16] takes advantage of the existence of this explicit expression for the density of optimally controlled trajectories $p_{u^*}$. PICE does not directly optimize the expected cost, instead it minimizes the KL-divergence $KL(p^*||p_{u_\theta})$ which measures the deviation of a parametrized distribution $p_{u_\theta}$ from the optimal one $p^*$. Although direct cost optimization and PICE are different methods, their global minimum is the same if the parametrization of $u_\theta$ can express the optimal control $u^* = u_{\theta^*}$. The parameters $\theta^*$ of the optimal controller are found using gradient descent:

$$\nabla_{\theta} KL(p^*||p_{u_\theta}) = \frac{1}{Z_{p_{u_\theta}}} \left\langle \exp \left( -\frac{1}{\gamma} S_{p_{u_\theta}}^\gamma (\tau) \right) \nabla_{\theta} \log p_{u_\theta}(\tau) \right\rangle_{p_{u_\theta}}, \quad (5.6)$$

where $Z_{p_{u_\theta}} := \left\langle \exp \left( -\frac{1}{\gamma} S_{p_{u_\theta}}^\gamma (\tau) \right) \right\rangle_{p_{u_\theta}}$.

That PICE uses the optimal density as a guidepost for the policy optimization might give it an advantage compared to direct cost-optimization. In practice however, this method only works properly if the initial guess of the controller $u_{\theta^*}$ does not deviate too much from the optimal control, as a high value of $KL(p^*||p_{u_{\theta}})$ leads to a high variance of the gradient estimator and results in bootstrapping problems of the algorithm [RK17, TGK16]. In the next section we introduce a method that interpolates between direct cost-optimization and the PICE method. This allows us to take advantage of the analytical solution of the optimal density without being hampered by the same bootstrapping problems as PICE.

\footnote{Note that this is not a trivial statement, as we now take the minimum also over non-Markovian processes with non-Gaussian noise.}
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.4 Interpolating between the two methods: Smoothing stochastic control problems

Cost function smoothing was recently introduced as a way to speed up optimization of neural networks [COO+17]: Optimization of a general cost function $f(\theta)$ can be speeded up by smoothing $f(\theta)$ using an inf-convolution with a distance kernel $d(\theta', \theta)^2$. The smoothed function

$$J^\alpha(\theta) = \inf_{\theta'} \alpha d(\theta', \theta) + f(\theta')$$

preserves the global minima of the function $f(\theta)$. To apply gradient descent based optimization on $J^\alpha(\theta)$ instead of $f(\theta)$ may significantly speed up convergence [COO+17].

We want to use this accelerative effect to find the optimal parametrization of the controller $u_\theta$. Therefore, we smooth the cost function $C(p_{u_\theta})$ as a function of the parameters $\theta$. As $C(p_{u_\theta}) = \langle V(\tau) \rangle_{p_{u_\theta}} + \gamma KL(p_{u_\theta} \parallel p_0)$ is a functional on the space of probability distributions $p_{u_\theta}$, the natural distance\(^3\) is the KL-divergence $KL(p_{u_{\theta'}} \parallel p_{u_\theta})$. So we replace

$$f(\theta) \rightarrow C(p_{u_\theta})$$
$$d(\theta', \theta) \rightarrow KL(p_{u_{\theta'}} \parallel p_{u_\theta})$$

and obtain the smoothed cost $J^\alpha(\theta)$ as

$$J^\alpha(\theta) = \inf_{\theta'} \alpha KL(p_{u_{\theta'}} \parallel p_{u_\theta}) + C(p_{u_{\theta'}})$$
$$= \inf_{\theta'} \alpha KL(p_{u_{\theta'}} \parallel p_{u_\theta}) + \gamma KL(p_{u_{\theta'}} \parallel p_0) + \langle V(\tau) \rangle_{p_{u_{\theta'}}}.$$  

Note the different roles of $\alpha$ and $\gamma$: $\alpha$ penalizes the deviation of $p_{u_{\theta'}}$ from $p_{u_\theta}$, while $\gamma$ penalizes the deviation of $p_{u_{\theta'}}$ from the uncontrolled dynamics $p_0$.

5.4.1 Computing the smoothed cost and its gradient

The smoothed cost $J^\alpha$ is expressed as a minimization problem that has to be solved. Here we show that for Path Integral control problems this can be done analytically. To do this we first show that we can replace $\inf_{\theta'} \rightarrow \inf_{p'}$ and then solve the minimization over $p'$ analytically. We replace the minimization over $\theta'$ by a minimization over $p'$ in two steps: first we state an assumption that allows us to

---

\(^2\)This is a generalized description. The authors in [COO+17] used $d(\theta', \theta) = |\theta' - \theta|^2$.

\(^3\)Remark: Strictly speaking the KL is not a distance, but a directed divergence.
5.4. Interpolating between the two methods: Smoothing stochastic control problems

replace \( \inf_{\theta'} \rightarrow \inf_{u'} \) and then proof that for Path Integral control problems we can replace \( \inf_{u'} \rightarrow \inf_{p'} \).

We assume that for every \( u_{\theta} \) and any \( \alpha > 0 \), the minimizer \( \theta^*_{a, \theta} \) over the parameter space

\[
\theta^*_{a, \theta} \coloneqq \arg \min_{\theta'} \alpha KL(p_{u_{\theta'}} || p_{u_\theta}) + C(p_{u_{\theta'}})
\]  

(5.9)
is the parametrization of the minimizer \( u^*_{a, \theta} \) over the function space

\[
u^*_{a, \theta} \coloneqq \arg \min_{u'} \alpha KL(p_{u'} || p_{u_\theta}) + C(p_{u'}),
\]
such that \( u^*_{a, \theta} \equiv u_{\theta^*_{a, \theta}} \). We call this assumption full parametrization. Naturally it is sufficient for full parametrization if \( u_{\theta}(x, t) \) is a universal function approximator with a fully observable state space \( x \) and the time \( t \) as input, although this may be difficult to achieve in practice. With this assumption we can replace \( \inf_{\theta'} \rightarrow \inf_{u'} \).

Analogously we replace \( \inf_{u'} \rightarrow \inf_{p'} \); in appendix 5.11 we proof that for Path Integral control problems the minimizer \( u^*_{a, \theta} \) over the function space induces the minimizer \( p^*_{a, \theta} \) over the space of probability distributions

\[
p^*_{a, \theta} \coloneqq \arg \min_{p'} \alpha KL(p' || p_{u_\theta}) + C(p'),
\]  

(5.10)
such that \( p^*_{a, \theta} \equiv p_{u^*_{a, \theta}} \). This step is similar to the derivation of of equation (5.5) in section 5.3.2, but now we have added an additional term \( \alpha KL(p_{u'} || p_{u_\theta}) \).

Hence, given a Path Integral control problem and a controller \( u_{\theta} \) that satisfies full parametrization we can replace \( \inf_{\theta'} \rightarrow \inf_{p'} \) and equation (5.8) becomes

\[
J^a(\theta) = \inf_{p'} \alpha KL(p' || p_{u_\theta}) + \gamma KL(p' || p_0) + \langle V(\tau) \rangle_{p'}.
\]  

(5.11)

This can be solved directly: first we compute the minimizer (see appendix 5.12 for details)

\[
p^*_{a, \theta}(\tau) = \frac{1}{Z_{p_{u_\theta}}^a} p_{u_\theta}(\tau) \exp \left( -\frac{1}{\gamma + \alpha} \gamma \theta (\tau) \right)
\]  

(5.12)

with the normalization constant \( Z_{p_{u_\theta}}^a = \langle \exp \left( -\frac{1}{\gamma + \alpha} \gamma \theta (\tau) \right) \rangle_{p_{u_\theta}} \). We plug this back in equation (5.11) and get an expression of the smoothed cost

\[
J^a(\theta) = -(\gamma + \alpha) \log \left( \exp \left( -\frac{1}{\gamma + \alpha} \gamma \theta (\tau) \right) \right)_{p_{u_\theta}}
\]  

(5.13)

and its gradient (for details see appendix 5.13)

\[
\nabla_{\theta} J^a(\theta) = -\frac{\alpha}{Z_{p_{u_\theta}}^a} \left\langle \exp \left( -\frac{1}{\gamma + \alpha} \gamma \theta (\tau) \right) \nabla_{\theta} \log p_{u_\theta} (\tau) \right\rangle_{p_{u_\theta}},
\]

(5.14)

which both can be estimated by samples from the distribution \( p_{u_\theta} \).
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.4.2 PICE, direct cost-optimization and risk sensitivity as limiting cases of smoothed cost optimization

The smoothed cost and its gradient depend on the two parameters \( \alpha \) and \( \gamma \), which come from the smoothing equation (5.7) and the definition of the control problem (5.2) respectively. Although at first glance the two parameters seem to play a similar role, they change different properties of the smoothed cost \( J^\alpha(\theta) \) when they are varied.

In the expression for the smoothed cost (5.13), the parameter \( \alpha \) only appears in the sum \( \gamma + \alpha \). Varying it changes the effect of the smoothing but leaves the optimum \( \theta^* = \arg\min_{\theta} J^\alpha(\theta) \) of the smoothed cost invariant (see appendix 5.14). We therefore call \( \alpha \) the smoothing parameter. The larger \( \alpha \), the weaker the smoothing; in the limiting case \( \alpha \to \infty \), smoothing is turned off as we can see from equation (5.13): for very large \( \alpha \), the exponential and the logarithmic function linearise, \( J^\alpha(\theta) \to C(p_{u_\theta}) \) and we recover direct cost-optimization. For the limiting case \( \alpha \to 0 \), we recover the PICE method: the optimizer \( p^*_{\alpha,\theta} \) becomes equal to the optimal density \( p^* \) and the gradient on the smoothed cost (5.14) becomes proportional to the PICE gradient (5.6):

\[
\lim_{\alpha \to 0} \frac{1}{\alpha} \nabla_\theta J^\alpha(\theta) = \nabla_\theta KL(p^*||p_{u_\theta}).
\]

Varying \( \gamma \) changes the control problem and thus its optimal solution. For \( \gamma \to 0 \), the control cost becomes zero. In this case the cost only consists of the state cost and arbitrary large controls are allowed. We get

\[
J^\alpha(\theta) = -\alpha \log \left\langle \exp\left( -\frac{1}{\alpha} V(\tau) \right) \right\rangle_{p_{u_\theta}}.
\]

This expression is identical to the risk sensitive control cost proposed in [FS02, FM95, BWK12]. Thus, for \( \gamma = 0 \), the smoothing parameter \( \alpha \) controls the risk-sensitivity, resulting in risk seeking objectives for \( \alpha > 0 \) and risk avoiding objectives for \( \alpha < 0 \). In the limiting case \( \gamma \to \infty \), the problem becomes trivial; the optimal controlled dynamics becomes equal to the uncontrolled dynamics: \( p^* \to p_0 \) (see (5.5)) and \( u^* \to 0 \).

If both parameters \( \alpha \) and \( \gamma \) are small, the problem is hard (see [RK17, TGK16]) as many samples are needed to estimate the smoothed cost. The problem becomes feasible if either \( \alpha \) or \( \gamma \) is increased. Increasing \( \gamma \) however, changes the control problem, while increasing \( \alpha \) weakens the effect of smoothing. In the remainder of this article we analyse, first theoretically in section 5.5 and then numerically in section 5.7, the effect that a finite \( \alpha > 0 \) has on the iterative optimization of the control \( u_{\theta} \) for a fixed value \( \gamma \).
5.5. The effect of cost function smoothing on policy optimization

We introduced smoothing as a way to speed up policy optimization compared to a direct optimization of the cost. In this section we analyse policy optimization with and without smoothing and show analytically how smoothing can speed up policy optimization. To simplify notation, we overload $p_{u_{\theta}} \rightarrow \theta$ so that we get $C(p_{u_{\theta}}) \rightarrow C(\theta)$ and $KL(p_{u_{\theta}}' || p_{u_{\theta}}) \rightarrow KL(\theta' || \theta)$.

We use a trust region constraint to robustly optimize the policy (compare [PMA10, SLA+15, GKPN14]). There are two options. On the one hand, we can directly optimize the cost $C$:

**Definition 1.** We define the direct update with stepsize $\varepsilon$ as an update $\theta \rightarrow \theta'$ with $\theta' = \Theta_{\varepsilon}^C(\theta)$ and

$$\Theta_{\varepsilon}^C(\theta) := \arg\min_{\theta'} C(\theta') \quad \text{s.t.} \quad KL(\theta' || \theta) \leq \varepsilon.$$  \hspace{1cm} (5.15)

The direct update results in the minimal cost that can be achieved after one single update. We define the optimal one-step cost

$$C_{\varepsilon}^*(\theta) := \min_{\theta'} C(\theta') \quad \text{s.t.} \quad KL(\theta' || \theta) \leq \varepsilon.$$  \hspace{1cm} (5.16)

On the other hand we can optimize the smoothed cost $J^\alpha$:

**Definition 2.** We define the smoothed update with stepsize $\varepsilon$ as an update $\theta \rightarrow \theta'$ with $\theta' = \Theta_{\varepsilon}^{J^\alpha}(\theta)$ and

$$\Theta_{\varepsilon}^{J^\alpha}(\theta) := \arg\min_{\theta'} J^\alpha(\theta') \quad \text{s.t.} \quad KL(\theta' || \theta) \leq \varepsilon.$$  \hspace{1cm} (5.16)

While a direct update achieves the minimal cost that can be achieved after a single update, we show below that a smoothed update can result in a faster cost reduction if more than one update step is performed.

**Definition 3.** We define the optimal two-step update $\theta \rightarrow \Theta' \rightarrow \Theta''$ as an update that results in the lowest cost that can be achieved with a two-step update $\theta \rightarrow \theta' \rightarrow \theta''$ with fixed stepsizes $\varepsilon$ and $\varepsilon'$ respectively:

$$\Theta', \Theta'' := \arg\min_{\theta', \theta''} C(\theta'') \quad \text{s.t.} \quad KL(\theta'' || \theta') \leq \varepsilon', \quad KL(\theta' || \theta) \leq \varepsilon$$
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization and the corresponding optimal two-step cost

\[ C^*_\mathcal{E},\mathcal{E}'(\theta) := \min_{\hat{\theta}'} \min_{\hat{\theta}''} C(\hat{\theta}'') \quad \text{s.t.} \quad KL(\hat{\theta}'\|\theta) \leq \mathcal{E}, \quad \text{s.t.} \quad KL(\hat{\theta}''\|\theta') \leq \mathcal{E}' \]

\[ = \min_{\hat{\theta}'} C\left(\Theta_{\mathcal{E}'}^{\hat{\theta}'}(\theta)\right) \quad \text{s.t.} \quad KL(\hat{\theta}'\|\theta) \leq \mathcal{E}. \quad (5.17) \]

In figure 5.1 we illustrate how such an optimal two-step update leads to a faster decrease of the cost than two consecutive direct updates.

**Theorem 1.** Statement 1: For all \( \mathcal{E} \), \( \alpha \) there exists an \( \mathcal{E}' \), such that a smoothed update with stepsize \( \mathcal{E} \) followed by a direct update with stepsize \( \mathcal{E}' \) is an optimal two-step update:

\[ \Theta' = \Theta_{\mathcal{E}}^{\hat{\theta}'}(\theta) \]

\[ \Theta'' = \Theta_{\mathcal{E}'}^{\hat{\theta}''}(\Theta') \]

\[ \Rightarrow C(\Theta'') = C^*_\mathcal{E},\mathcal{E}'(\theta) \]

The size of the second step \( \mathcal{E}' \) is a function of \( \theta \) and \( \alpha \). Statement 2: \( \mathcal{E}' \) is monotonically decreasing in \( \alpha \).

While it is evident from equation (5.17) that the second step of the optimal two-step update must be a direct update, the statement that the first step is a smoothed update is non-trivial. We proof this and statement 2 in appendix 5.16.

Direct updates are myopic and do not take into account successive steps and are thus suboptimal when more than one update is needed. Smoothed updates on the other hand, as we see on theorem 1, anticipate a subsequent step and minimize the cost that results from this this two-step update. Hence smoothed updates favour a greater cost reduction in the future over maximal cost reduction in the current step. The strength of this anticipatory effect depends on the smoothing strength, which is controlled by the smoothing parameter \( \alpha \): For large \( \alpha \), smoothing is weak and the size \( \mathcal{E}' \) of this anticipated second step becomes small. Figure 5.1 B illustrates that for this case, when \( \mathcal{E}' \) becomes small, smoothed updates become more similar to direct updates. In the limiting case \( \alpha \to \infty \) the difference between smoothed and direct updates vanishes completely, as \( J^{\alpha}(\theta) \to C(\theta) \) (see section 5.4.2).

We expect that also with multiple update steps due to this anticipatory effect, iterating smoothed updates leads to a faster decrease of the cost than iterating direct updates. We will confirm this by numerical studies. Furthermore, we expect that this accelerating effect of smoothing is stronger for smaller values of \( \alpha \). On the other hand, as we will discuss in the next section, for smaller values of \( \alpha \) it
is harder to accurately perform the smoothed updates. Therefore we expect an optimal performance for an intermediate value of $\alpha$. Based on this we build an algorithm in the next section that aims to accelerate policy optimization by cost function smoothing.

## 5.6 Numerical Method

In this section we develop an algorithm that takes a parametrized control function $u_\theta$ with initial parameters $\theta_0$ and updates these parameters in each iteration $n$ using smoothed updates.

### 5.6.1 Compute smoothed and direct updates using natural gradients

So far we have specified the smoothed updates $\theta_{n+1} = \Theta^{J^\alpha}_E(\theta_n)$ (5.16) in an abstract manner and left open how to perform this optimization step. To compute an explicit expression we introduce a Lagrange multiplier $\beta$ and express the constraint optimization (5.16) as an unconstrained optimization

$$
\theta_{n+1} = \arg\min_{\theta'} J^\alpha(\theta') + \beta KL(\theta'||\theta_n) \tag{5.18}
$$

Following [SLA+15] we assume that the trust region size $\mathcal{E}$ is small. For small $\mathcal{E} \ll 1$ we get $\beta \gg 1$ and can expand $J^\alpha(\theta')$ to first and $KL(\theta'||\theta_n)$ to second order (see appendix 5.15 for the details). This gives

$$
\theta_{n+1} = \theta_n - \beta^{-1} F^{-1} \nabla_{\theta'} J^\alpha(\theta')|_{\theta' = \theta_n}, \tag{5.19}
$$

a natural gradient update with the Fisher-matrix $F = \nabla_{\theta'} \nabla_{\theta}^T KL(\theta'||\theta_n)|_{\theta' = \theta_n}$ (we use the conjugate gradient method to approximately compute the natural gradient for high dimensional parameter spaces. See appendix 5.19 or [SLA+15] for details). The parameter $\beta$ is determined using a line search such that$^4$

$$
KL(\theta_n||\theta_{n+1}) = \mathcal{E}.
$$

Note that for direct updates this derivation is the same, just replace $J^\alpha$ by $C$.

$^4$For practical reasons, we reverse the arguments of the KL, since it is easier to estimate it from samples drawn from the first argument. For very small values, the $KL$ is approximately symmetric in its arguments.
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.6.2 Reliable gradient estimation using adaptive smoothing

To compute smoothed updates using equation (5.19) we need the gradient of the smoothed cost. We assume full parametrization and use equation (5.14), which can be estimated using weighted samples drawn from the distribution $p_{\theta}$:

$$\nabla_{\theta} J^\alpha(\theta) \approx \alpha \sum_{i=1}^{N} w^i \log p_{u_{\theta}}(\tau^i). \quad (5.20)$$

The weights are given by

$$w^i = \frac{1}{Z} \exp\left(-\frac{1}{\gamma + \alpha} S^\gamma_{p_{u_{\theta}}}(\tau^i)\right),$$

with the normalization constant $Z = \sum_{i=1}^{N} \exp\left(-\frac{1}{\gamma + \alpha} S^\gamma_{p_{u_{\theta}}}(\tau^i)\right)$.

The variance of this estimator depends sensitively on the entropy of the weights $H_N(w) = -\sum_{i=1}^{N} w^i \log(w^i)$.

If the entropy is low, the total weight is concentrated on a few particles. This results in a poor gradient estimator where only a few of the particles actually contribute. This concentration is dependent on the smoothing parameter $\alpha$: for small $\alpha$, the weights are very concentrated in a few samples, resulting in a large weight-entropy and thus a high variance of the gradient estimator. As small $\alpha$ corresponds to strong smoothing, we want $\alpha$ to be as small as possible, but large enough to allow a reliable gradient estimation. Therefore, we set a bound to the weight entropy $H_N(w)$. To get a bound that is independent of the number of samples $N$, we use that in the limit of $N \to \infty$ the weight entropy is monotonically related to the KL-Divergence $KL(p_{\alpha,u_{\theta}}^* || p_{u_{\theta}})$

$$KL(p_{\alpha,u_{\theta}}^* || p_{u_{\theta}}) = \lim_{N \to \infty} \log N - H_N(w)$$

(see appendix 5.18). This provides a method for choosing $\alpha$ independently of the number of samples: we set the constraint $KL(p_{\alpha,u_{\theta}}^* || p_{u_{\theta}}) \leq \Delta$ and determine the smallest $\alpha$ that satisfies this condition by using a line search. Large values of $\Delta$ correspond to small values of $\alpha$ (see appendix 5.17) and therefore strong smoothing, we thus call $\Delta$ the smoothing strength.

5.6.3 Formulating a model free algorithm

We can compute the gradient (5.20) and the KL-divergence while treating the dynamical system as a black-box. For this we write the probability distribution $p_{u_{\theta}}$
over trajectories $\tau$ as a Markov process:

$$p_{u_\theta}(\tau) = \prod_{0 < t < T} p_{u_\theta}(x_{t+dt}|x_t, t),$$

where the product runs over the time $t$, which is discretized with time step $dt$. We define the noisy action $a_t = u(x_t, t) + \xi_t$ and formulate the Markov transitions $p_{u_\theta}(x_{t+dt}|x_t)$ for the dynamical system (5.1) as

$$p_{u_\theta}(x_{t+dt}|x_t) = \delta(x_{t+dt} - F(x_t, a_t, t)) \cdot \pi_\theta(a_t|t, x_t),$$

with $\delta(\cdot)$ the dirac delta function. This splits the transitions up into the deterministic dynamical system $F(x_t, a_t, t)$ and a gaussian policy $\pi_\theta(a_t|t, x_t) = \mathcal{N}(a_t | u_\theta(x_t, t), \nu dt)$ with mean $u_\theta(x_t, t)$ and variance $\nu dt$. Using this we get a simplified expression for the gradient of the smoothed cost (5.20) that is independent of the system dynamics, given the samples drawn from the controlled system $p_{u_\theta}$:

$$\nabla_\theta J_\alpha(\theta) \approx \alpha N \sum_{i=1}^{N} \sum_{0 < t < T} w_i \nabla_\theta \log \pi_\theta(a^i_t|t, x^i_t).$$

Similarly we obtain an expression for the estimator of the KL divergence

$$KL(\theta_n || \theta_{n+1}) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{0 < t < T} \log \frac{\pi_{\theta_\alpha}(a^i_t|t, x^i_t)}{\pi_{\theta_{n+1}}(a^i_t|t, x^i_t)}.$$  

With this we formulate ASPIC (Algorithm 2) which optimizes the parametrized policy $\pi_\theta$ by iteratively drawing samples from the controlled system. The algorithm does not depend on the specific dynamics of the system, and is therefore model free.

## 5.7 Numerical Experiments

In a series of numerical experiments we compare the convergence speed of policy optimization with and without smoothing and find that smoothing accelerates convergence. For the optimization with smoothing, we use ASPIC (Algorithm 2) and for the optimization without smoothing, we use a version of ASPIC where we replaced the gradient of the smoothed cost with the gradient of the cost itself. For a simple linear quadratic control problem we analyse the convergence speed for different values of the smoothing strength $\Delta$. Further details about the numerical experiments are found in appendix 5.21.

---

5 When using the Euler method to integrate the dynamical system we get $F(x_t, a_t) = (x_t + dt \cdot (f(x_t, t) + g(x_t, t)a_t))$. 

133
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.7.1 A simple LQ control problem: Brownian Viapoints

We first consider a simple task of controlling a Brownian particle in 1D

\[ \dot{x} = u(x, t) + \xi. \]  

(5.21)

We define the state cost as a quadratic penalty for deviating from the targets \( x_i \) at the different times \( t_i \): \( V(x, t) = \sum_i \delta(t - t_i) \frac{(x - x_i)^2}{2\sigma^2} \) with \( \sigma = 0.1 \). As a parametrized controller we use a time varying linear feedback controller, i.e. \( u_\theta(x, t) = \theta_{1,t} x + \theta_{0,t} \). This controller fulfils the requirement of full parametrization for this task (see appendix 5.20). For further details of the numerical experiment see appendix 5.21.1.

We apply ASPIC to this control problem and compare its performance for different sizes of the smoothing strength \( \Delta \) (see figure 5.2). The results confirm our expectations from the discussions in the theory and method sections (see sections 5.5 and 5.6.2). As predicted by theory we observe an acceleration of the policy optimization when smoothing is switched on. This acceleration becomes more pronounced when \( \Delta \) is increased, which we attribute to an increase of the anticipatory effect of the smoothed updates as smoothing becomes stronger (see section 5.5). When \( \Delta \) is too large the performance of the algorithm deteriorates again, which is in line with our discussion of gradient estimation problems that arise for strong smoothing (see section 5.6.2).

5.7.2 Nonlinear control problems

We tested the performance of the algorithm on three non-linear control problems: the acrobot swing-up task, the pendulum swing-up task and a two dimensional walker task. The 2D walker task was simulated using the OpenAI gym [BCP+16]. For the acrobot and pendulum swing-up tasks we used a time-varying linear feedback controller, for the 2D walker task we parametrized the control \( u_\theta \) using a neural network. Further details of the three tasks are found in appendix 5.21.2 for the pendulum, 5.21.3 for the acrobot and 5.21.4 for the 2DWalker.

In all three tasks smoothing improves the convergence rate of policy optimization (see figures 5.3, 5.4, 5.5). So for all tasks, including the LQ control problem (figure 5.2), ASPIC requires less iterations to achieve the same cost reduction as direct cost-optimization, with only a negligible amount of additional computational steps that do not depend on the complexity of the simulation runs. For the non-linear control task the parametrized controller does not strictly meet the requirement of full parametrization that we need to derive the gradient of the smoothed cost. Nevertheless, the tasks show that a strong performance boost can also be achieved, when this full parametrization assumption is violated.
Figure 5.1: Illustration of optimal two-step updates compared with two consecutive direct updates. Illustrated is a two-dimensional cost landscape $C(\theta)$ parametrized by $\theta$. Dark colors represent low cost, while light colors represent high cost. Green dots indicate the optimal two-step update $\theta \rightarrow \Theta' \rightarrow \Theta''$ while red dots indicate two consecutive direct updates $\theta \rightarrow \theta' \rightarrow \theta''$ with $\theta' = \Theta_C^C(\theta)$ and $\theta'' = \Theta_C^C(\theta')$. The dashed circles indicate trust regions. $\theta'$, $\theta''$ and $\Theta''$ are the minimizers of the cost in the trust regions around $\theta$, $\theta'$ and $\Theta'$ respectively. $\Theta'$ is chosen such that the cost $C(\Theta'')$ after the subsequent direct update is minimized. In both panels, the final cost after an optimal two-step update $C(\Theta'')$ is smaller than the final cost after two direct updates $C(\theta'')$. In panel A) the sizes of the update steps are equal $\delta = \delta'$. In panel B) it is illustrated, that when the size of the second step becomes small $\delta' \ll \delta$ the smoothed update $\theta \rightarrow \Theta'$ becomes more similar to the direct update $\theta \rightarrow \theta'$. 
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

Algorithm 2 ASPIC - Adaptive Smoothing of Path Integral Control

Require: State cost function $V(x, t)$
- control cost parameter $\gamma$
- base policy that defines uncontrolled dynamics $\pi_0$
- simulator of system dynamics with a parametrized policy $\pi_\theta$
- trust region sizes $\mathcal{E}$
- smoothing strength $\Delta$
- number of samples $N$

initialize $\theta_0$
$n = 0$
repeat
  draw samples $\tau^i$, with $i = 1, \ldots, N$, from simulator controlled by parametrized policy $\pi_{\theta_n}$
  for each sample $i$ compute $S^\gamma_{\pi_{\theta_n}}(\tau^i) = \sum_{0 < t < T} V(x^i_t, t) + \gamma \log \frac{\pi_{\theta_n}(a^i_t | t, x^i_t)}{\pi_0(a^i_t | t, x^i_t)}$
  \{Find minimal $\alpha$ such that $KL \leq \Delta$\}
  $\alpha \leftarrow 0$
  repeat
    increase $\alpha$
    $S^i_\alpha \leftarrow S^\gamma_{\pi_{\theta_n}}(\tau^i) \cdot \frac{1}{\gamma + \alpha}$
    compute weights $w_i \leftarrow \exp(-S^i_\alpha)$
    normalize weights $w_i \leftarrow \frac{w_i}{\sum_i (w_i)}$
    compute sample size independent weight entropy $KL \leftarrow \log N + \sum_i w_i \log(w_i)$
  until $KL \leq \Delta$
  \{whiten the weights\}
  $\hat{w}_i \leftarrow \frac{w_i - \text{mean}(w_i)}{\text{std}(w_i)}$
  \{compute the gradient on the smoothed cost\}
  $g \leftarrow \sum_i \sum_t \hat{w}_i \frac{\partial}{\partial \theta} \log \pi_\theta(a^i_t | t, x^i_t)|_{\theta = \theta_n}$
  \{compute Fisher matrix\}
  use conjugate gradient descent to compute an approximate solution to the natural gradient $g_F = F^{-1} g$ (see appendix 5.19)
  do line search to compute step size $\eta$ such $KL(\theta_n || \theta_{n+1}) = \mathcal{E}$.
  update parameters $\theta_{n+1} \leftarrow \theta_n + \eta \cdot g_F$
  $n = n + 1$
until convergence
5.7. Numerical Experiments

Figure 5.2: LQ control problem: Viapoints. For each iteration we used $N = 100$ rollouts to compute the gradient. A) Plotted is the number of iterations needed for the cost to cross a threshold $C \leq 2 \cdot 10^4$ versus the smoothing strength $\Delta$. For $\Delta = 0$ there is no smoothing. Increasing the smoothing strength results in a faster decrease of the cost; when $\Delta$ is increased further the performance decreases again. Shown is the mean and the standard deviation over 10 runs of the algorithm. B) Plotted is the cost versus the iterations of the algorithm. Direct optimization of the cost exhibits a slower convergence rate than optimization of the smoothed cost with $\Delta = 0.2 \log 100$. 

```python
# Example code

def plot_fignum(filename):
    pass
```
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

Figure 5.3: Pendulum swing-up task. For each iteration we used $N = 500$ rollouts to compute the gradient. Plotted is the cost versus the iterations of the algorithm for an optimization of the smoothed cost with a smoothing strength $\Delta = 0.5$ and for a direct cost-optimization. Shown are the mean and the standard deviation for 10 runs of the algorithm. Smoothed cost-optimization exhibits a faster convergence rate than direct cost-optimization.
Figure 5.4: Acrobot swing-up task. For each iteration we used $N = 500$ rollouts to compute the gradient. Plotted is the cost versus the iterations of the algorithm for an optimization of the smoothed cost with a smoothing strength $\Delta = 0.5$ and for a direct cost-optimization. Shown are the mean and the standard deviation for 10 runs of the algorithm. Smoothed cost-optimization exhibits a faster convergence rate than direct cost-optimization.
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

Figure 5.5: 2D-walker task. For each iteration we used $N = 100$ rollouts to compute the gradient. Plotted is the cost versus the iterations of the algorithm for an optimization of the smoothed cost with a smoothing strength $\Delta = 0.05 \log 100$ and for a direct cost-optimization. Shown are the mean and the standard deviation for 10 runs of the algorithm. Smoothed cost-optimization exhibits a faster convergence rate than direct cost-optimization.
5.8 Discussion

For Path Integral control problems the optimal control policy can serve as a guidepost for policy optimization. This is used in the PICE algorithm [KR16]. One might hope that a representation of optimal control can help to find a parametrized policy and surpass the more general approach of direct cost-optimization. In practice however, the PICE algorithm suffers from problems with sample efficiency [RK17]. We introduced a smoothing technique for the cost functions of Path Integral control problems that allows to interpolate between direct cost-optimization and the PICE method. In between these extremes we have found a method that is superior to direct cost-optimization while remaining feasible.

We conducted a theoretical analysis of the optimization of smoothed cost-functions and showed that minimizing the smoothed cost can accelerate policy optimization by having less myopic updates that favour stronger cost reduction in subsequent updates over immediate cost reduction in the current step. This prediction is confirmed by our numerical experiments, which show that smoothing the cost accelerates the convergence of policy optimization. While the theoretical analysis only makes statements for optimizations with a total of two update steps, the numerical experiments show that the acceleration effect persists when more than two update steps are performed.

Because direct cost-optimization and the PICE method are recovered in the limits of weak and strong smoothing respectively, we examined smoothed cost optimization for different levels of smoothing. The result shows in both limits the performance of the algorithm deteriorates. For weak smoothing this can be explained with the disappearance of the accelerating effect that is caused by smoothing. The deterioration of performance for strong smoothing may be attributed to gradient estimation problems which also appear in PICE [RK17]. These gradient estimation problems only appear only for strong smoothing, while the accelerative effect stays noticeable when smoothing is weak.

The explanatory power of the theoretical results regarding the numerical experiments is limited through the fact that our derivation of the smoothed cost and its gradient requires an assumption on the representational power of the parametrized control policy. In principal, a universal function approximator, like an infinitely large neural network, would be sufficient to fulfil this full parametrization assumption. However in practice, where we have to rely on function approximators with a finite number of parameters this is difficult to obtain. Nevertheless, the qualitative behaviour, that smoothing speeds up policy optimization, persists despite this deviation of the numerical methods from the theoretical assumptions.

To conduct the numerical studies we used the algorithm ASPIC that we developed based on our theoretical results. ASPIC uses robust updates and an adaptive adjustment of the smoothing parameter to ensure that the gradient on the smoothed cost stays computable with a finite amount of samples. This procedure bears simi-
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

Assigning smoothing to an adaptive annealing scheme, with the smoothing parameter playing the role of an artificial temperature. However in contrast to classical annealing schemes, such as simulated annealing, changing the smoothing parameter does not change the optimization target: the minimum of the smoothed cost remains the optimal control solution for all levels of smoothing.

In the weak smoothing limit, ASPIC directly optimizes the cost using trust region constrained updates, similar to the TRPO algorithm [SLA+15]. TRPO differs from ASPIC’s weak smoothing limit by additionally using certain variance reduction techniques for the gradient estimator: They replace the stochastic cost in the gradient estimator by the easier-to-estimate advantage function, which has a state dependent baseline and only takes into account future expected cost. Since this depends on the linearity of the gradient in the stochastic cost and this dependence is non-linear for the gradient of the smoothed cost, we cannot directly incorporate these variance reduction techniques in ASPIC.

In the strong smoothing limit ASPIC becomes a version of PICE [KR16] that—unlike the plain PICE algorithm—uses a trust region constraint to achieve robust updates. The gradient estimation problem that appears in the PICE algorithm was previously addressed in [RK17]: they proposed a heuristic that allows to reduce the variance of the gradient estimator by adjusting the particle weights used to compute the policy gradient. In [RK17] this heuristic is introduced as an ad hoc fix of the sampling problem and the adjustment of the weights introduces a bias with possible unknown side effects. Our study sheds a new light on this, as adjusting the particle weights corresponds to a change of the smoothing parameter in our case. The theoretical results we derived can however not directly be transferred to [RK17], as [RK17] does not use trust regions to bound the updates of the policy optimization, which we assumed for our analysis.

Especially when samples are expensive to compute it is important to squeeze out as much information from them as possible. We showed that for Path Integral control problems a smoothed version of the cost function and its gradient can directly be computed from the samples and allows to make less myopic policy updates than cost-greedy methods (like TRPO and PIREPS) and thereby accelerate convergence. To fully benefit from this, it is important future work to develop variance reduction techniques for the gradient of the smoothed cost similar to the techniques already used for methods that directly optimize the cost. A possible way to achieve this would be control variates that are tailored to the gradient estimator of the smoothed cost [RGB14,Gla13]. Another important future work is to develop a deeper understanding of the full parametrization assumption and how its violation impacts the performance of the algorithm. Minimizing this impact might be an important lever to boost the performance of policy optimization for Path Integral control problems.
5.9 Supplement: Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.10 Derivation of the policy gradient

Here we derive equation (5.4). We write \( C(p_{u\theta}) = \left\langle S_{u\theta}^\gamma (\tau) \right\rangle_{p_{u\theta}} \), with \( S_{u\theta}^\gamma (\tau) := V(\tau) + \gamma \log \frac{p_{u\theta}(\tau)}{p_0(\tau)} \) and take the derivative of equation (5.2):

\[
\nabla_\theta \left\langle S_{u\theta}^\gamma (\tau) \right\rangle_{p_{u\theta}} = \nabla_\theta \left\langle V(\tau) + \gamma \log \frac{p_{u\theta}(\tau)}{p_0(\tau)} \right\rangle_{p_{u\theta}} \tag{5.22}
\]

Now we introduce the importance sampler \( p_{u\theta'} \) and correct for it.

\[
\nabla_\theta \left\langle S_{u\theta}^\gamma (\tau) \right\rangle_{p_{u\theta}} = \nabla_\theta \left\langle \frac{p_{u\theta}(\tau)}{p_{u\theta'}(\tau)} \left( V(\tau) + \gamma \log \frac{p_{u\theta}(\tau)}{p_0(\tau)} \right) \right\rangle_{p_{u\theta'}} \tag{5.23}
\]

This is true for all \( \theta' \) as long as \( p_{u\theta}(\tau) \) and \( p_{u\theta'}(\tau) \) are absolutely continuous to each other. Taking the derivative we get:

\[
= \left\langle \nabla_\theta \frac{p_{u\theta}(\tau)}{p_{u\theta'}(\tau)} \left( V(\tau) + \gamma \log \frac{p_{u\theta}(\tau)}{p_0(\tau)} \right) \right\rangle_{p_{u\theta'}} + \left\langle \frac{p_{u\theta}(\tau)}{p_{u\theta'}(\tau)} \left( \gamma \frac{1}{p_{u\theta}(\tau)} \nabla_\theta p_{u\theta}(\tau) \right) \right\rangle_{p_{u\theta'}} \tag{5.24}
\]

\[
= \left\langle \nabla_\theta \log p_{u\theta}(\tau) \right\rangle_{p_{u\theta}} \left( V(\tau) + \gamma \log \frac{p_{u\theta}(\tau)}{p_0(\tau)} \right) + \gamma \nabla_\theta \left\langle \frac{1}{p_{u\theta'}(\tau)} \right\rangle_{p_{u\theta'}} \tag{5.25}
\]

\[
= \left\langle S_{u\theta}^\gamma(\tau) \nabla_\theta \log p_{u\theta}(\tau) \right\rangle_{p_{u\theta}} + \gamma \nabla_\theta \left\langle 1 \right\rangle_{p_{u\theta}} \tag{5.26}
\]

\[
= \left\langle S_{u\theta}^\gamma(\tau) \nabla_\theta \log p_{u\theta}(\tau) \right\rangle_{p_{u\theta}} \tag{5.27}
\]

5.11 Replacing minimization over \( u \) by minimization over \( p' \)

Here we show that for

\[
J^u(\theta) = \inf_{u'} \alpha KL(p_{u'} || p_{u\theta}) + \gamma KL(p_{u'} || p_0) + \langle V(\tau) \rangle_{p'} \tag{5.28}
\]
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

we can replace the minimization over $u$ by a minimization over $p'$ to obtain equation (5.11). For this, we need to show that the minimizer $p^*_{\alpha,\theta}$ of equation (5.11) is induced by $u^*_{\alpha,\theta}$, the minimizer of equation (5.28):

$$p^*_{\alpha,\theta} \equiv p_{u^*_{\alpha,\theta}}.$$  

The solution to (5.11) is given by (see appendix 5.12)

$$p^*_{\alpha,\theta} = \frac{1}{Z} p_{u_\theta}(\tau) \exp \left( -\frac{1}{\gamma + \alpha} S_\tau^{\gamma} p_{u_\theta}(\tau) \right)$$  \hspace{1cm} (5.29)

$$= \frac{1}{Z} p_{u_\theta}(\tau) \left( \frac{p_\theta(\tau)}{p_{u_\theta}(\tau)} \right)^{\frac{\gamma}{\gamma + \alpha}} \exp \left( -\frac{1}{\gamma + \alpha} V(\tau) \right).$$  \hspace{1cm} (5.30)

We rewrite

$$p_0(\tau) \left( \frac{p_{u_\theta}(\tau)}{p_\theta(\tau)} \right)^{1 - \frac{\gamma}{\gamma + \alpha}} =$$

$$p_0(\tau) \exp \left( \left( 1 - \frac{\gamma}{\gamma + \alpha} \right) \int_0^T \left( \frac{1}{2} u_\theta(x_t, t)^T u_\theta(x_t, t) + u_\theta(x_t, t)^T \xi_t \right) dt \right)$$

where we used the Girsanov theorem [BK14, TK15] (and set $\nu = 1$ for simpler notation). With $\tilde{u}_\theta(x_t, t) := (1 - \frac{\gamma}{\gamma + \alpha}) u_\theta(x_t, t)$ this gives

$$p_0(\tau) \left( \frac{p_{u_\theta}(\tau)}{p_\theta(\tau)} \right)^{1 - \frac{\gamma}{\gamma + \alpha}} = p_0(\tau) \exp \left( \int_0^T \left( \frac{1}{2} \tilde{u}_\theta(x_t, t)^T \tilde{u}_\theta(x_t, t) + \tilde{u}_\theta(x_t, t)^T \xi_t \right) dt \right) \cdot$$

$$\cdot \exp \left( \left( \frac{1}{2} \alpha \tilde{u}_\theta(x_t, t)^T \tilde{u}_\theta(x_t, t) \right) dt \right)$$

$$= p_{\tilde{u}_\theta}(\tau) \exp \left( \int_0^T \left( \frac{1}{2} \alpha \tilde{u}_\theta(x_t, t)^T \tilde{u}_\theta(x_t, t) \right) dt \right)$$

So we get

$$p^*_{\alpha,\theta} = \frac{1}{Z} p_{\tilde{u}_\theta}(\tau) \exp \left( \int_0^T \left( \frac{1}{2} \alpha \tilde{u}_\theta(x_t, t)^T \tilde{u}_\theta(x_t, t) \right) dt \right) \exp \left( -\frac{1}{\gamma + \alpha} V(\tau) \right).$$  \hspace{1cm} (5.31)

This has the form of an optimally controlled distribution with dynamics

$$\dot{x}_t = f(x_t, t) + g(x_t, t) (\tilde{u}_\theta(x_t, t) + \hat{u}(x_t, t) + \xi_t)$$  \hspace{1cm} (5.32)
and cost
\[\left\langle \int_0^T \frac{1}{\gamma + \alpha} V(x_t, t) - \frac{1}{2}\alpha \hat{u}\theta(x_t, t)^T \hat{u}\theta(x_t, t) dt + \int_0^T \left( \frac{1}{2} \hat{u}(x_t, t)^T \hat{u}(x_t, t) + \hat{u}(x_t, t)^T \xi_t \right) dt \right\rangle_{p_\theta} \]

This is a Path Integral control problem with state cost \(\int_0^T \frac{1}{\gamma + \alpha} V(x_t, t) - \frac{1}{2\alpha} \hat{u}\theta(x_t, t)^T \hat{u}\theta(x_t, t) dt\) which is well defined with \(\hat{u}\theta(x_t, t) = (1 - \frac{1}{\gamma + \alpha})u\theta(x_t, t)\).

Let \(\hat{u}\) be the optimal control of this Path Integral control problem. Then \(p^*_{\alpha, \theta}\) is induced by equation (5.32) with \(\hat{u} = \hat{u}\). This is equivalent to say that \(p^*_{\alpha, \theta}\) is induced by equation (5.1) As \(p^*_{\alpha, \theta}\) is the density that minimizes equation (5.11), \(\hat{u}\theta + \hat{u}\) is minimizing equation (5.28).

### 5.12 Minimizer of smoothed cost

Here we want to proof equation (5.12):

\[p^*_{\alpha, \theta}(\tau) := \arg \min_{p'} aKL(p'\|p_{u\theta}) + \left\langle S^\gamma_{p_{u\theta}}(\tau) \right\rangle_{p'} \]

\[= \arg \min_{p'} \left\langle \alpha \log \frac{p'(\tau)}{p_{u\theta}(\tau)} + V(\tau) + \gamma \log \frac{p'(\tau)}{p_0(\tau)} + \kappa \right\rangle_{p'} \]

\[0 = \frac{\delta}{\delta p'(\tau)} \left\langle \alpha \log \frac{p'(\tau)}{p_{u\theta}(\tau)} + V(\tau) + \gamma \log \frac{p'(\tau)}{p_0(\tau)} + \kappa \right\rangle_{p' = p^*_{\alpha, \theta}} \]

where we added a Lagrange multiplier \(\kappa\) to ensure normalization. We get

\[0 = \alpha \log \frac{p'(\tau)}{p_{u\theta}(\tau)} + V(\tau) + \gamma \log \frac{p'(\tau)}{p_0(\tau)} + \kappa \]

\[= \exp \left( \frac{\kappa}{\alpha + \gamma} \right) p_{u\theta}(\tau)^{\frac{\alpha}{\alpha + \gamma}} p_0(\tau)^{\frac{\gamma}{\alpha + \gamma}} \exp \left( -\frac{1}{\gamma + \alpha} V(\tau) \right) \]

\[= \exp \left( \frac{\kappa}{\alpha + \gamma} \right) p_{u\theta}(\tau) \exp \left( -\frac{1}{\gamma + \alpha} V(\tau) - \frac{\gamma}{\alpha + \gamma} \log \frac{p_{u\theta}(\tau)}{p_0(\tau)} \right) \]

\[= \exp \left( \frac{\kappa}{\alpha + \gamma} \right) p_{u\theta}(\tau) \exp \left( -\frac{1}{\gamma + \alpha} S^\gamma_{p_{u\theta}}(\tau) \right) \]

where \(\kappa\) is chosen such that the distribution is normalized.
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.13 Derivation of the gradient of the smoothed cost function

Here we derive equation (5.14) by taking the derivative of equation (5.13):

\[
\nabla_{\theta} J^\alpha(\theta) = - (\gamma + \alpha) \nabla_{\theta} \log \left\langle \exp \left(- \frac{1}{\gamma + \alpha} \left(V(\tau) + \gamma \log \frac{p_{u_\theta}(\tau)}{p_0(\tau)}\right)\right) \right\rangle_{p_{u_\theta}} \\
= - \frac{\gamma + \alpha}{Z_{p_{u_\theta}}} \nabla_{\theta} \left\langle \exp \left(- \frac{1}{\gamma + \alpha} \left(V(\tau) + \gamma \log \frac{p_{u_\theta}(\tau)}{p_0(\tau)}\right)\right) \right\rangle_{p_{u_\theta}} \\
(5.41)
\]

Now we introduce the importance sampler \( p_{u_\theta'} \) and correct for it.

\[
\nabla_{\theta} J^\alpha(\theta) = - \frac{\gamma + \alpha}{Z_{p_{u_\theta}}} \nabla_{\theta} \left\langle \frac{p_{u_\theta}(\tau)}{p_{u_\theta'}(\tau)} \exp \left(- \frac{1}{\gamma + \alpha} \left(V(\tau) + \gamma \log \frac{p_{u_\theta}(\tau)}{p_0(\tau)}\right)\right) \right\rangle_{p_{u_\theta'}} \\
= - \frac{\gamma + \alpha}{Z_{p_{u_\theta}}} \nabla_{\theta} \left\langle \frac{p_0(\tau)^{\gamma+\alpha}}{p_{u_\theta'}(\tau)} \left(p_{u_\theta}(\tau)\right)^{\gamma+\alpha} \exp \left(- \frac{1}{\gamma + \alpha} V(\tau)\right) \right\rangle_{p_{u_\theta'}} \\
= - \frac{\gamma + \alpha}{Z_{p_{u_\theta}}} \nabla_{\theta} \left\langle \frac{1}{p_{u_\theta'}(\tau)} \left(p_{u_\theta}(\tau)\right)^{\gamma+\alpha} \exp \left(- \frac{1}{\gamma + \alpha} V(\tau)\right) \nabla_{p_{u_\theta}} \right\rangle_{p_{u_\theta'}} \\
= - \frac{\gamma + \alpha}{Z_{p_{u_\theta}}} \nabla_{\theta} \left\langle \exp \left(- \frac{1}{\gamma + \alpha} S_{p_{u_\theta}}(\tau)\right) \nabla_{\theta} \log p_{u_\theta}(\tau) \right\rangle_{p_{u_\theta'}} \\
(5.42)
\]

5.14 Global maximum preserved under full parametrization

Here we show that smoothing leaves the global optimum of the cost \( C(p_{u_\theta}) \) invariant.

**Proof.** As \( KL(p_{u_\theta'}||p_{u_\theta}) \geq 0 \) we have that

\[
J^\alpha(\theta) = \inf_{\theta'} C(p_{u_\theta'}) + \alpha KL(p_{u_\theta'}||p_{u_\theta}) \geq \inf_{\theta'} C(p_{u_\theta'}) = C(p_{u_\theta'})
\]

To show that the global minimum \( \theta^* \) of \( C \) is also the global minimum of \( J^\alpha \), it is thus sufficient to show that

\[
J^\alpha(\theta^*) \leq C(p_{u_\theta'})
\]
5.15 Smoothed updates for small update steps $\varepsilon$

We have

$$J^\varepsilon(\theta^*) = \inf_{\theta'} C(p_{u_{\theta'}}) + \alpha KL(p_{u_{\theta'}} || p_{u_{\theta^*}})$$

Using that the minimum of a sum of terms is never larger than the sum of the minimum of terms, we get

$$J^\varepsilon(\theta^*) \leq \left(\inf_{\theta'} C(p_{u_{\theta'}})\right) + \left(\inf_{\theta'} \alpha KL(p_{u_{\theta'}} || p_{u_{\theta^*}})\right)$$

$$= C(p_{u_{\theta^*}}) + \alpha KL(p_{u_{\theta^*}} || p_{u_{\theta^*}})$$

$$= C(p_{u_{\theta^*}})$$

\[\square\]

5.15 Smoothed updates for small update steps $\varepsilon$

We want to compute equation (5.18) for small $\varepsilon$ which corresponds to large $\beta$. Assuming a smooth dependence of $p_{u_{\theta}}$ on $\theta$, bounding $KL(\theta||\theta_n)$ to a very small value allows us to do a Taylor expansion which we truncate at second order:

$$\arg\min_{\theta'} J^\varepsilon(\theta') + \beta KL(\theta'||\theta_n) \approx (5.47)$$

$$\approx \arg\min_{\theta'} (\theta' - \theta_n)^T \nabla_{\theta'} J^\varepsilon(\theta') + \frac{1}{2}(\theta' - \theta_n)^T (H + \beta F)(\theta' - \theta_n)$$

$$= \theta_n - \beta^{-1} F^{-1} \nabla_{\theta'} J^\varepsilon(\theta') \bigg|_{\theta'=\theta_n} + O(\beta^{-2})$$

with

$$H = \nabla_{\theta'} \nabla_{\theta'}^T J^\varepsilon(\theta') \bigg|_{\theta'=\theta_n}$$

$$F = \nabla_{\theta'} \nabla_{\theta'}^T KL(\theta'||\theta_n) \bigg|_{\theta'=\theta_n}.$$

See also [Mar14]. We used that $\varepsilon \ll 1 \iff \beta \gg 1$. With this the Fisher information $F$ dominates over the Hessian $H$ and thus the Hessian does not appear anymore in the update equation. This defines a natural gradient update with stepsize $\beta^{-1}$

5.16 Smoothing Theorem

Here we proof theorem 1. We split the proof of theorem 1 into three subsections: In the first subsection we state and proof a lemma that we need to proof statement 1, in the second subsection we proof statement 1 and in the third subsection we proof statement 2.
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.16.1 Lemma

**Lemma 1.** With $\theta^*_a, \theta$ defined as in equation (5.9) and $\mathcal{E}_a(\theta) = KL(\theta^*_a || \theta)$ we can rewrite $J^a(\theta)$:

$$J^a(\theta) = C(\Theta^C_{\mathcal{E}_a(\theta)}(\theta))_{\mathcal{E}'=\mathcal{E}_a(\theta)} + \alpha \mathcal{E}_a(\theta)$$  \hspace{1cm} (5.50)

**Proof.** With the definition of $\theta^*_a, \theta$ as the minimizer of $C(\theta') + \alpha KL(\theta'||\theta)$ (see (5.9)) we have

$$J^a(\theta) = C(\theta^*_a, \theta) + \alpha KL(\theta^*_a || \theta) = C(\theta^*_a) + \alpha \mathcal{E}_a(\theta).$$

What is left to show is that

$$\theta^*_a, \theta \equiv \Theta^C_{\mathcal{E}_a(\theta)}(\theta).$$

As $\Theta^C_{\mathcal{E}_a(\theta)}(\theta)$ is the minimizer of the cost $C$ within the trust region defined by $\{\theta' : KL(\theta'||\theta) \leq \mathcal{E}_a(\theta)\}$ we have to show that

1. $\theta^*_a, \theta$ lies within in this trust region,

2. $C(\theta^*_a, \theta)$ is a minimizer of the cost $C$ within this trust region.

The first point is trivially true as $KL(\theta^*_a, \theta) = \mathcal{E}_a(\theta)$ by definition. Hence $\theta^*_a, \theta$ lies at the boundary of this trust region and therefore in it, as the boundary belongs to the trust region. The second point we proof by contradiction: Given $\theta^*_a, \theta$ is not minimizing the cost within the trust region, then there exists a $\hat{\theta}$ with $C(\hat{\theta}) < C(\theta^*_a, \theta)$ and $KL(\hat{\theta} || \theta) \leq \mathcal{E}_a(\theta) = KL(\theta^*_a || \theta)$. Therefore it must hold that

$$C(\hat{\theta}) + \alpha KL(\hat{\theta} || \theta) < C(\theta^*_a, \theta) + \alpha KL(\theta^*_a, \theta)$$

which is a contradiction, as $\theta^*_a, \theta$ is the minimizer of $C(\theta') + \alpha KL(\theta' || \theta)$. \hfill \Box

5.16.2 Proof of statement 1

Here we show that for every $\alpha$ and $\theta$ there exists an $\mathcal{E}' = \mathcal{E}_a^*(\theta)$ such that

$$C(\Theta^C_{\mathcal{E}}(\Theta^a_{\mathcal{E}'}(\theta)))_{\mathcal{E}'=\mathcal{E}_a^*(\theta)} = C^*_a, \mathcal{E}' \bigg|_{\mathcal{E}'=\mathcal{E}_a^*(\theta)}.$$  \hspace{1cm} (5.51)

**Proof.** As $J^a(\theta)$ is the infimum of $C(\theta') + \alpha KL(\theta' || \theta)$, we have for any $\mathcal{E}' > 0$

$$J^a(\theta) \leq C(\Theta^C_{\mathcal{E}'}(\theta)) + \alpha KL(\Theta^C_{\mathcal{E}'}(\theta || \theta))$$
Further, as \( \Theta^C_{\theta'}(\theta) \) lies in the trust region \( \{ \theta' : KL(\theta'\|\theta) \leq \varepsilon' \} \) we have that \( KL(\Theta^C_{\theta'}(\theta)\|\theta) \leq \varepsilon' \), so we can write
\[
C(\Theta^C_{\theta'}(\theta)) + aKL(\Theta^C_{\theta'}(\theta)\|\theta) \leq C(\Theta^C_{\theta'}(\theta)) + a\varepsilon'
\]
and thus
\[
J^a(\theta) \leq C(\Theta^C_{\theta'}(\theta)) + a\varepsilon'.
\]
Next we minimize both sides of this inequality within the trust region \( \{ \theta' : KL(\theta'\|\theta) \leq \varepsilon \} \).
We use that
\[
J^a(\Theta^J_{\theta'}(\theta)) = \min_{\theta'} J^a(\theta')
\]
and get
\[
J^a(\Theta^J_{\theta'}(\theta)) \leq \min_{\theta'} \left( C(\Theta^C_{\theta'}(\theta')) + a\varepsilon' \right).
\]
(5.52)
Now we use lemma 1 and rewrite the left hand side of this inequality.
\[
J^a(\Theta^J_{\theta'}(\theta)) = C(\Theta^C_{\theta'}(\Theta^J_{\theta'}(\theta)))|_{\theta' = \theta} + a\varepsilon_a(\theta)
\]
with \( \varepsilon_a(\theta) := \varepsilon_a(\Theta^J_{\theta'}(\theta)) \). Plugging this back to (5.52) we get
\[
C(\Theta^C_{\theta'}(\Theta^J_{\theta'}(\theta)))|_{\theta' = \theta} + a\varepsilon_a(\theta) \leq \min_{\theta'} \left( C(\Theta^C_{\theta'}(\theta')) + a\varepsilon' \right).
\]
As this inequality holds for any \( \varepsilon' > 0 \) we can plug in \( \varepsilon_a(\theta) \) on the right hand side
of this inequality and obtain
\[
C(\Theta^C_{\theta'}(\Theta^J_{\theta'}(\theta)))|_{\theta' = \theta} + a\varepsilon_a(\theta) \leq \min_{\theta'} C(\Theta^C_{\theta'}(\theta'))|_{\theta' = \theta} + a\varepsilon_a(\theta).
\]
We subtract \( a\varepsilon_a(\theta) \) on both sides
\[
C(\Theta^C_{\theta'}(\Theta^J_{\theta'}(\theta)))|_{\theta' = \theta} \leq \min_{\theta'} C(\Theta^C_{\theta'}(\theta'))|_{\theta' = \theta}.
\]
Using equation (5.17) gives
\[
C(\Theta^C_{\theta'}(\Theta^J_{\theta'}(\theta)))|_{\theta' = \theta} \leq C^*_C(\theta)|_{\theta' = \theta}.
\]
which concludes the proof.
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

5.16.3 Proof of statement 2

Here we show that $\mathcal{E}' = \mathcal{E}'_a(\theta)$ is a monotonically decreasing function of $\alpha$. $\mathcal{E}'_a(\theta)$ is given by

$$ \mathcal{E}'_a(\theta) = \mathcal{E}_a(\Theta_{\mathcal{E}}^n(\theta)) = KL(\theta_{a,\theta'}^* || \theta') \bigg|_{0^* = R_{\mathcal{E}}(\theta)}.$$  

We have

$$\left( \alpha KL(\theta_{a,\theta'}^* || \theta') + C(\theta_{a,\theta'}^*) \right) \bigg|_{\theta'=R_{\mathcal{E}}(\theta)} = \left( \inf_{\theta''} \alpha KL(\theta'' || \theta') + C(\theta'') \right) \bigg|_{\theta'=R_{\mathcal{E}}(\theta)} = \min_{\theta'' \text{ s.t. } KL(\theta'' || \theta) \leq \mathcal{E}} \inf_{\theta''} \alpha KL(\theta'' || \theta') + C(\theta'').$$

For convenience we introduce a shorthand notation for the minimizers

$$\theta_a := \Theta_{\mathcal{E}}^n(\theta),$$
$$\theta'_a := \theta_{a,\theta'}^* \big|_{\theta='R_{\mathcal{E}}(\theta)}.$$

We compare $\alpha_1 \geq 0$ with $\mathcal{E}'_{a_1}(\theta) := KL(\theta_{a_1}' || \theta_{a_1})$ and $\alpha_2 \geq 0$ with $\mathcal{E}'_{a_2}(\theta) := KL(\theta_{a_2}' || \theta_{a_2})$ and assume that $\mathcal{E}'_{a_1}(\theta) < \mathcal{E}'_{a_2}(\theta)$. We show that from this it follows that $\alpha_1 > \alpha_2$.

**Proof.** As $\theta'_{a_1}, \theta'_{a_2}$ minimize $\alpha_1 KL(\theta' || \theta) + C(\theta')$ we have

$$\alpha_1 KL(\theta'_{a_1} || \theta_{a_1}) + C(\theta'_{a_1}) \leq \alpha_1 KL(\theta'_{a_2} || \theta_{a_2}) + C(\theta'_{a_2})$$
$$\Rightarrow \alpha_1 \mathcal{E}_{a_1}(\theta) + C(\theta'_{a_1}) \leq \alpha_1 \mathcal{E}_{a_2}(\theta) + C(\theta'_{a_2})$$

and analogous for $\alpha_2$

$$\alpha_2 KL(\theta'_{a_1} || \theta_{a_1}) + C(\theta'_{a_1}) \geq \alpha_2 KL(\theta'_{a_2} || \theta_{a_2}) + C(\theta'_{a_2})$$
$$\Rightarrow \alpha_2 \mathcal{E}_{a_1}(\theta) + C(\theta'_{a_1}) \geq \alpha_2 \mathcal{E}_{a_2}(\theta) + C(\theta'_{a_2})$$

With $\mathcal{E}_{a_1}(\theta) < \mathcal{E}_{a_2}(\theta)$ we get

$$\alpha_1 \geq \frac{C(\theta'_{a_1}) - C(\theta'_{a_2})}{\mathcal{E}_{a_2}(\theta) - \mathcal{E}_{a_1}(\theta)} \geq \alpha_2.$$

We showed that from $\mathcal{E}_{a_1}(\theta) < \mathcal{E}_{a_2}(\theta)$ it follows that $\alpha_1 \geq \alpha_2$ which proofs that $\mathcal{E}_a(\theta)$ is monotonously decreasing in $\alpha$. 

150
5.17 \( \Delta \) is monotonic in \( \alpha \)

Now we show that

\[
\Delta = KL(p_{\alpha,\theta}^*||p_{u_\theta})
\]

is a monotonic function of \( \alpha \).

\[
\frac{\partial}{\partial \alpha} KL(p_{\alpha,\theta}^*||p_{u_\theta}) = \frac{\partial}{\partial \alpha} \left\langle \ln \frac{p_{\alpha,\theta}^*}{p_{u_\theta}} \right\rangle_{p_{u_\theta}^*} = \frac{\partial}{\partial \alpha} \left\langle \left( \frac{\partial}{\partial \alpha} \frac{p_{\alpha,\theta}^*}{p_{u_\theta}} \right) \ln \frac{p_{\alpha,\theta}^*}{p_{u_\theta}} \right\rangle_{p_{u_\theta}^*} + \left\langle \frac{p_{\alpha,\theta}^*}{p_{u_\theta}} \frac{\partial}{\partial \alpha} \ln \frac{p_{\alpha,\theta}^*}{p_{u_\theta}} \right\rangle_{p_{u_\theta}^*} = \left\langle \left( \frac{\partial}{\partial \alpha} \frac{p_{\alpha,\theta}^*}{p_{u_\theta}} \right) \ln \frac{p_{\alpha,\theta}^*}{p_{u_\theta}} \right\rangle_{p_{u_\theta}^*} + 1 \frac{\partial}{\partial \alpha} p_{\alpha,\theta}^* \frac{p_{\alpha,\theta}^*}{p_{u_\theta}}
\]

\[
\frac{\partial}{\partial \alpha} Z_{p_{u_\theta}}^\alpha = \exp \left( -\frac{1}{\gamma + \alpha} S_{p_{u_\theta}}^\gamma (\tau) \right)
\]

we get

\[
\frac{\partial}{\partial \alpha} p_{\alpha,\theta}^* = \frac{1}{(\gamma + \alpha)^2} S_{p_{u_\theta}}^\gamma (\tau) \frac{p_{\alpha,\theta}^*}{p_{u_\theta}} \frac{1}{p_{u_\theta}} Z_{p_{u_\theta}}^\alpha \frac{\partial}{\partial \alpha} Z_{p_{u_\theta}}^\alpha
\]

\[
\frac{\partial}{\partial \alpha} Z_{p_{u_\theta}}^\alpha = \left\langle \frac{1}{(\gamma + \alpha)^2} S_{p_{u_\theta}}^\gamma \exp \left( -\frac{1}{\gamma + \alpha} S_{p_{u_\theta}}^\gamma (\tau) \right) \right\rangle_{p_{u_\theta}}.
\]
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization.

and thus

$$\frac{\partial}{\partial \alpha} p^*_{a,\theta} = \frac{1}{(\gamma + \alpha)^2} S^\gamma_{p_{u\theta}}(\tau) \frac{p^*_{a,\theta}}{p_{u\theta}} - \frac{1}{(\gamma + \alpha)^2} \left( \frac{S^\gamma_{p_{u\theta}}}{p^*_{a,\theta}} \right)$$

So finally we get

$$\frac{\partial}{\partial \alpha} KL(p^*_{a,\theta} \| p_{u\theta}) = \frac{1}{(\gamma + \alpha)^2} \left( \frac{p^*_{a,\theta}}{p_{u\theta}} \left( S^\gamma_{p_{u\theta}}(\tau) - \left< S^\gamma_{p_{u\theta}} \right>_{p^*_{a,\theta}} \right) \ln \frac{p^*_{a,\theta}}{p_{u\theta}} \right)$$

So

$$\Delta = KL(p^*_{a,\theta} \| p_{u\theta})$$

is a monotonically decreasing function of $\alpha$.

5.18 Proof for equivalence of weight entropy and KL-divergence

We want to show that

$$\lim_{N \to \infty} \log N - H_N(w) = \lim_{N \to \infty} \log N + \sum_{i=1}^N w_i \log(w_i)$$

$$= KL(p^*_{a,\theta} \| p_{u\theta}).$$

Where the samples $i$ are drawn from $p_{u\theta}$ and the $w^i$ are given by

$$w^i = \frac{1}{\sum_i^N \exp\left( -\frac{1}{\gamma + \alpha} S_{p_{u\theta}}(\tau^i) \right)} \exp\left( -\frac{1}{\gamma + \alpha} S_{p_{u\theta}}(\tau^i) \right),$$
5.18. Proof for equivalence of weight entropy and KL-divergence

We get

\[ \lim_{N \to \infty} \log N + \sum_{i=1}^{N} w^i \log(w^i) = \]

\[ = \lim_{N \to \infty} \log N + \sum_{i=1}^{N} \frac{1}{\sum_{i}^{N}} \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau^i)} \right) \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau^i)} \right) \cdot \log \left( \frac{1}{\sum_{i}^{N}} \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau^i)} \right) \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau^i)} \right) \right) \]

\[ = \lim_{N \to \infty} \log N + \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\sum_{i}^{N}} \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau^i)} \right) \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau^i)} \right) \cdot \log \left( \frac{1}{\sum_{i}^{N}} \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau^i)} \right) \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau^i)} \right) \right) \]

Now we replace in the limit \( N \to \infty \), \( \frac{1}{N} \sum_{i}^{N} \to \langle p_{ug} \rangle : \)

\[ = \left\langle \frac{1}{\exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau)} \right)_{p_{ug}}} \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau)} \right) \cdot \log \left( \frac{1}{\exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau)} \right)_{p_{ug}}} \exp\left( -\frac{1}{\gamma + \alpha S_{p_{ug}}^Y (\tau)} \right) \right) \right\rangle_{p_{ug}} \]
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

Using equation (5.12) this gives

\[
\begin{align*}
&= \log \left( \frac{1}{\exp \left( \frac{1}{\gamma + \alpha} S_{p_{u_0}}^T \right) p_{u_0}} \exp \left( - \frac{1}{\gamma + \alpha} S_{p_{u_0}}^T \left( \tau \right) \right) \right) \\
&= \log \left( \frac{1}{\exp \left( \frac{1}{\gamma + \alpha} S_{p_{u_0}}^T \right) p_{u_0}} \exp \left( - \frac{1}{\gamma + \alpha} S_{p_{u_0}}^T \left( \tau \right) \right) \right) \\
&= KL(p_{a,\theta}^* || p_{u_0})
\end{align*}
\]

5.19 Inversion of the Fisher matrix

We compute an approximation to the natural gradient \( g_f = F^{-1} g \) by approximately solving the linear equation \( F g_f = g \) using truncated conjugate gradient. With the normal gradient \( g \) and the Fisher matrix \( F = \nabla_\theta \nabla_\theta^T KL(p_{u_0} || p_{u_0}) \) (see appendix 5.15).

We use an efficient way to compute the Fisher vector product \( F y \) [SLA+15] using an automated differentiation package: First for each rollout \( i \) and timepoint \( t \) the symbolic expression for the gradient on the KL multiplied by a vector \( y \) is computed:

\[
a_{i,t}(\theta_{n+1}) = \left( \nabla_{\theta_{n+1}}^T \log \frac{\pi_{\theta_n}(a_{i}^t|t,x_{i}^t)}{\pi_{\theta_{n+1}}(a_{i}^t|t,x_{i}^t)} \right) y
\]

Then we take the second derivative on this scalar quantity, sum over all times and average over the samples. This gives then the Fisher vector

\[
F y = \frac{1}{N} \sum_{i=1}^{N} \sum_{0 \leq t < T} \nabla_{\theta_{n+1}} a_{i,t}(\theta_{n+1}).
\]

5.20 Full parametrization in LQ problem

Here we discuss why for a linear quadratic problem a time varying linear controller is a full parametrization. We want to show that for every

\[
p_{a,\theta_0}^* = \frac{1}{Z} p_{u_0}(\tau) \exp \left( - \frac{1}{\gamma + \alpha} S_{p_{u_0}}^T (\tau) \right) \tag{5.53}
\]
there is a time varying linear controller $u_{\theta_0}$ such that $p_{u_{\theta_0}} = p_{\alpha_0}^*$. We assume that $u_{\theta_0}$ is a time varying linear controller. In appendix 5.11 we have shown that $u_{\alpha_0}^*$ is the solution to the Path Integral control problem with dynamics

$$
\dot{x}_t = f(x_t, t) + g(x_t, t) (\tilde{u}(x_t, t) + \hat{u}(x_t, t) + \xi_t)
$$

and cost

$$
\left\langle \int_0^T \frac{1}{\gamma} V(x_t, t) - \frac{1}{2} \gamma \tilde{u}(x_t, t)^T \tilde{u}(x_t, t) dt + \int_0^T \left( \frac{1}{2} \hat{u}(x_t, t)^T \hat{u}(x_t, t) + \hat{x}_t(x_t, t)^T \xi_t \right) dt \right\rangle_{p_{\alpha}}
$$

with $\tilde{u} = (1 - \frac{\gamma}{\gamma + \alpha}) u_{\theta_0}(x_t, t)$.

It is now easy to see that if $u_{\theta_0}$ is a time varying linear controller, thus a linear function of the state, the cost is a quadratic function of the state $x$ (note that $V(x_t, t)$ is quadratic in the LQ case). Thus for all values of $\alpha$, $u_{\alpha_0}^*$ is the solution to a linear quadratic control problem and thus a time varying linear controller (see e.g. [KS72]). Therefore a time varying linear controller is a full parametrization.

5.21 Details for the numerical experiments

5.21.1 LQ-control

Dynamics:

- The dynamics are ODEs integrated by an Euler scheme.
- Dynamic equations: see section 5.7.1.
- The differential equation is initialized at $x = 0$.
- $dt = 0.1$

Control problem:

- $\gamma = 1$.
- Time-Horizon $T = 10s$
- State-Cost function: see section 5.7.1. $(x_0, t_0) = (-10, 1), (x_1, t_1) = (10, 2), (x_2, t_2) = (-10, 3), (x_3, t_3) = (-20, 4), (x_4, t_4) = (-100, 5), (x_5, t_5) = (-50, 6), (x_6, t_6) = (10, 7), (x_7, t_7) = (20, 8), (x_8, t_8) = (30, 9)$
- Variance of uncontrolled dynamics $\nu = 1$
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

Algorithm:

- Batchsize: $N = 100$
- $\varepsilon = 0.1$
- $\Delta = 0.2\log 100$
- Conjugate gradient iterations: 2 (for each time step separately)

The parametrized controller was initialized at $\theta = 0$.

5.21.2 Pendulum

The differential equation for the pendulum is:

$$\ddot{x} + c\omega_0\dot{x} + \omega_0^2 \sin(x) = \lambda (u + \xi)$$

with

- $c\omega_0 = 0.1 [s^{-1}]$
- $\omega_0^2 = 10. [s^{-2}]$
- $\lambda = 0.2$

We implemented this differential equation as a first order differential equation and integrated it with an Euler scheme with $dt = 0.01$. The pendulum is initialized resting at the bottom:

$$\dot{x} = 0, x = 0.$$ 

As a parametrized controller we use a time varying linear feedback controller:

$$u_\theta(x, \dot{x}, t) = \theta_{3,t}\cos(x) + \theta_{2,t}\sin(x) + \theta_{1,t}\dot{x} + \theta_{0,t}.$$ 

The parametrized controller was initialized at $\theta = 0$.

Control-problem:

- $\gamma = 1$
- Time-Horizon: $T = 3.0s$
- The State-Cost function has End-Cost only:

$$V(x, \dot{x}, t) = \delta(t - T)(-500Y + 10\dot{x}^2)$$

with $Y = -\cos(x)$ (height of tip).

- Variance of uncontrolled dynamics $\nu = 1$
5.21. Details for the numerical experiments

Algorithm:

- Batchsize: $N = 500$
- $\varepsilon = 0.01$
- $\Delta = 0.5$
- The Fisher-matrix was inverted for each time step separately using the scipy pseudo-inverse with rcond=1e-4.

5.21.3 Acrobot

**Dynamics** We use the definition of the acrobot as in [1]. The differential equations for the acrobot are:

$$
\begin{align*}
d_{11}(x)\ddot{x}_1 + d_{12}(x)\ddot{x}_2 + h_1(x, \dot{x}) + \phi_1(x) &= 0 \\
d_{21}(x)\ddot{x}_1 + d_{22}(x, \dot{x}) + h_2(x, \dot{x}) + \phi_2(x) &= \lambda \cdot (u + \xi)
\end{align*}
$$

with

- $d_{11} = m_1l_{c1}^2 + m_2\left(l_1^2 + l_{c2}^2 + 2l_1l_{c2}\cos(x_2)\right) + I_1 + I_2$
- $d_{12} = m_2\left(l_{c2}^2 + l_1l_{c2}\cos(x_2)\right) + I_2$
- $d_{21} = d_{12}$
- $d_{22} = m_2l_{c2}^2 + I_2$
- $h_1 = -m_2l_1l_{c2}\sin(x_2)\left(\dot{x}_2^2 + 2\dot{x}_1\dot{x}_2\right)$
- $h_2 = m_2l_1l_{c2}\sin(x_2)\dot{x}_1^2$
- $\phi_2 = m_2l_{c2}G\cos(x_1 + x_2)$
- $\phi_1 = (m_1l_{c1} + m_2l_1)g\cos(x_1) + \phi_2$

with the parameter values

- $G = 9.8$
- $l_1 = 1. \text{ [m]}$
- $l_2 = 2. \text{ [m]}$
- $m_1 = 1. \text{ [kg]}$ mass of link 1
- $m_2 = 1. \text{ [kg]}$ mass of link 2
- $l_{c1} = 0.5 \text{ [m]}$ position of the center of mass of link 1
5. Cost function smoothing in Path Integral control problems accelerates convergence of gradient based policy optimization

- $l_{c2} = 1.0$ [m] position of the center of mass of link 2
- $I_1 = 0.083$ moments of inertia for both links
- $I_2 = 0.33$ moments of inertia for both links
- $\lambda = 0.2$

We implemented this differential equation as a first order differential equation and integrated it with an Euler scheme with $dt = 0.01$. The acrobot is initialized resting at the bottom:

$$\dot{x}_1 = 0, \dot{x}_2 = 0, x_1 = -\frac{1}{2}\pi, x_2 = 0.$$ 

As a parametrized controller we use a time varying linear feedback controller:

$$u_\theta(x, \dot{x}, t) = \theta_{8,t} \cos(x_1) + \theta_{7,t} \sin(x_2) + \theta_{6,t} \cos(x_2) + \theta_{5,t} \sin(x_2) +$$

$$+ \theta_{4,t} \sin(x_1 + x_2) + \theta_{3,t} \cos(x_1 + x_2) + \theta_{2,t} \dot{x}_1 + \theta_{1,t} \dot{x}_2 + \theta_{0,t}.$$ 

The parametrized controller was initialized at $\theta = 0$.

Control-problem:

- $\gamma = 1$.
- Time-Horizon: $T = 3.0s$
- The State-Cost function has End-Cost only:

$$V(x, \dot{x}, t) = \delta(t - T)(-500Y + 10(\dot{x}_1^2 + \dot{x}_2^2))$$

with $Y = -l_1 \cos(x_1) - l_2 \cos(x_1 + x_2)$ (height of tip).
- Variance of uncontrolled dynamics $\nu = 1$

Algorithm:

- Batchsize: $N = 500$
- $\epsilon = 0.01$
- $\Delta = 0.5$
- The Fisher-matrix was inverted for each time step separately using the scipy pseudo-inverse with rcond=$1e^{-4}$. 

158
5.21.4 Walker

For dynamics and the state cost function we used "BipedalWalker-v2" from the OpenAI gym [BCP+16]. The policy was a Gaussian policy, with static variance $\sigma = 1$. The state dependent mean of the Gaussian policy was a neural network controller with two hidden layers with 32 neurons, each. The activation function is a tanh. For the initialization we used Glorot Uniform (see [GB10]). The inputs to the neural network was the observation space provided by OpenAI gym task "BipedalWalker-v2": State consists of hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with ground, and 10 lidar rangefinder measurements.

**Control-problem:**

- $\gamma = 0$
- Time-Horizon: defined by OpenAI gym task "BipedalWalker-v2"
- State-Cost function defined by OpenAI gym task "BipedalWalker-v2": Reward is given for moving forward, total 300+ points up to the far end. If the robot falls, it gets -100. Applying motor torque costs a small amount of points, more optimal agent will get better score.

**Algorithm:**

- Batchsize: $N = 100$
- $\epsilon = 0.01$
- $\Delta = 0.05 \log 100$
- Conjugate gradient iterations: 10
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


171


Michael P Sceniak and M Bruce Maciver. Slow gaba(a) mediated synaptic transmission in rat visual cortex. BMC Neurosci, 9:8, 2008.


BIBLIOGRAPHY


Summary

In this work we studied three aspects of complex systems: self-organization, learning and control. We discovered a new paradigm of self-organization, broadened the scope of a known learning paradigm so that is can be used for modelling learning in the brain, extended the usage of a known control algorithm and introduced a new spectrum of control algorithms that unifies and accelerates several known algorithms.

In chapter 2 we looked at self-organization in single cells and addressed the question how proteins form pattern that are adapted to cell geometry. Inspired by an Escherichia Coli mutant in which a single protein could establish a polar pattern, we developed a protein interaction network that consists of a single protein species that can be in different states. The dynamics of these protein densities can be formulated by reaction diffusion equations, that are partial differential equations of the protein densities. Numerical solutions of these model-equations show that a protein with this interaction structure leads to a polar pattern in an elongated cell. In a series of numerical experiments we narrowed down the geometrical properties of the cell that influence the shape of the protein pattern. From this we concluded that the local ratio of membrane to bulk is the geometrical cue the pattern adapts to. Since the protein interaction network contains only generic chemical interactions, we assume that this new patterning mechanism could be the basis of many pattern formation phenomena in nature.

In chapter 3 we addressed the question if the FORCE learning-method for reservoir computing forms a possible mechanism for learning in the brain. It was an open question whether the this method, which was introduced for neurons that communicate with continuous signals, can also be applied to more realistic models of neurons that communicate via spikes, since spike-coding networks usually have a much more chaotic dynamic than networks which use a continuous coding. Our approach to this problem was to consider special neurons that use a precise spike coding scheme, as it can be shown that these precisely coding neural networks encode a continuous dynamical system. Previous work has introduced a precisely spiking neural network architecture that encodes a linear dynamical system. To obtain a suitable neural network that can be trained with the FORCE method we
extended this work to obtain a network of spiking neurons that encodes a nonlinear dynamical system. We demonstrated that this neural network can be trained to learn periodic and chaotic time series as well as memory-based tasks with an accuracy that was previously not possible for spike based neural networks. Further we extended the FORCE method to be able to learn the reaction of a dynamical system to white noise input. The input of white noise makes it difficult for the FORCE method to converge because the output-signal—an approximation of the target signal—that is fed back into the neural network is very erratic. We solved this problem by adding the the target signal to the network output before feeding it back. This allows the FORCE method to learn the dynamical system. With this method we trained a neural network to learn a model of a mechanical pendulum and demonstrated that this model can be used to compute a control strategy that allows to invert the pendulum. This broad variety of tasks that spiking networks can learn using the FORCE method makes FORCE an interesting candidate in the search for the principles of how the brain learns.

In chapter 4 we showed how to control the topology of a growing network. We demonstrated that a sampling based approach allows to achieve this despite the curse of dimensionality. For this we framed the problem as a KL-control problem in which the optimal cost-to-go is an expected value and can thus be estimated by sampling. Based on the formal solution to the optimal control problem we developed a parametrized importance sampler that has the functional form of the optimal control. We proposed an adaptive importance sampling algorithm that adapts these parameters such that the importance sampler approximates the optimal control, which is an optimal sampler. This allowed us to efficiently compute the optimal cost-to-go of the KL-control problem. We have illustrated the effectiveness of our method on the task of influencing the growth of conversation cascades in an internet forum and showed that this procedure leads to a good control strategy even if some of the assumptions that define the KL-control problem are violated.

In chapter 5 we introduced and studied a smoothing technique for the cost function of Path Integral control problems that leaves the global optimum invariant. Gradient descent on this smoothed cost function defines a spectrum of algorithms which contains greedy cost-optimization for zero smoothing and the recently introduced PICE method for infinitely strong smoothing. In contrast to PICE however the gradient on the smoothed cost function can be efficiently estimated from samples if the smoothing strength is not too large. We considered natural gradient descent and formulated a theorem that allows to understand the effect of smoothing on the cost optimization process and provided an argument for why smoothing can speed up optimization. We successfully tested these theoretical results using numerical experiments that show that natural gradient descent on the smoothed cost function is superior to greedy optimization of the un-smoothed cost and, in contrast to PICE, remains feasible.
Samenvatting

In dit werk hebben we drie aspecten van complexe systemen bestudeerd: zelforganisatie, leren en aansturing. Wij ontdekten een nieuw paradigma van zelforganisatie, verbreidden het bereik van een bekend leerparadigma, zodat het kan worden gebruikt voor het modelleren van het leren in de hersenen, hebben het gebruik van een bekend aansturingsalgoritme uitgebreid en een nieuw spectrum aan aansturingsalgoritmen geïntroduceerd die verschillende bekende algo-ritmen verenigt en versnelt.

In hoofdstuk 2 hebben we gekeken naar zelforganisatie in enkele cellen en hebben we de vraag behandeld hoe eiwitten een patroon vormen dat is aangepast aan de celgeometrie. Geïnspireerd door een Escherichia Coli-mutant waarin een enkel eiwit een polair patroon kon vormen, hebben we een eiwitinteractienetwerk ontwikkeld dat bestaat uit een enkele eiwit species die verschillende toestanden kan aanwenden. De dynamica van deze eiwitdichtheden kan worden geformuleerd door middel van reactiediffusievergelijkingen, die partiele differentiaalvergelijkingen zijn van de eiwitdichtheden. Numerieke oplossingen van deze modelvergelijkingen laten zien dat een eiwit met deze interactiestructuur leidt tot een polair patroon in een langwerpige cel. In een reeks numerieke experimenten ingrenzen we de geometrische eigenschappen van de cel die de vorm van het eiwitpatroon beïnvloeden. Hieruit concludeerden we dat de lokale verhouding van membraan tot bulk de geometrische cue is waar het patroon zich aan aanpast. Omdat het eiwitinteractienetwerk alleen generieke chemische interacties bevat, nemen we aan dat dit nieuwe patroonvormingsmechanisme de basis kan zijn van veel patroonvormingsverschijnselen in de natuur.

In hoofdstuk 3 hebben we de vraag behandeld of de FORCE leermethode voor reservoir computing een mogelijk mechanisme vormt om in de hersenen te leren. Het was een open vraag of de methode, die werd geïntroduceerd voor neuronen die communiceren met continue signalen, ook kan worden toegepast op meer realistische modellen van neuronen die via pulsen communiceren, omdat pulse-coderende netwerken meestal een veel chaotischere dynamiek hebben dan netwerken die een continue codering gebruiken. We hebben het probleem benaderd door rekening te houden met speciale neuronen die een nauwkeurig pulse-coderingsschema ge-
bruiken, omdat kan worden aangetoond dat deze nauwkeurig coderende neurale netwerken coderen voor een continu dynamisch systeem. In het vorige werk is een nauwkeurig vormgegeven neurale netwerkarchitectuur geïntroduceerd die codeert voor een lineair dynamisch systeem. Om een neuraal netwerk te krijgen dat geschikt is voor training met de FORCE-methode, hebben we dit werk uitgebreid om een netwerk van gepulseerde-neuronen te verkrijgen dat codeert voor een niet-lineair dynamisch systeem. We hebben aangetoond dat dit neurale netwerk kan worden getraind om periodieke en chaotische tijdreeksen te leren, evenals op geheugen gebaseerde taken met een nauwkeurigheid die voorheen niet mogelijk was voor pulse-gebaseerde neurale netwerken. Verder hebben we de FORCE-methode uitgebreid om de reactie van een dynamisch systeem op de invloed van witte ruis te leren. De invloed van witte ruis maakt het moeilijk voor de FORCE-methode om te convergeren omdat het uitgangssignaal een benadering van het doelsignaal dat wordt teruggevoerd naar het neurale netwerk zeer grillig is. We hebben dit probleem opgelost door het doelsignaal aan de netwerkuitvoer toe te voegen voordat het terug wordt gevoerd. Hierdoor kan de FORCE-methode het dynamische systeem leren. Met deze methode hebben we een neuraal netwerk getraind om een model van een mechanische slinger te leren en hebben we aangetoond dat dit model kan worden gebruikt om een aansturingsstrategie te berekenen waarmee de slinger kan worden omgekeerd. Deze hoeveelheid van taken die gepulseerde netwerken kunnen leren met de FORCE-methode maakt FORCE een interessante kandidaat bij het vinden van de principes van hoe de hersenen leren.

In hoofdstuk 4 hebben we laten zien hoe de topologie van een groeiend netwerk beheerd kan worden. We hebben aangetoond dat een op sampling gebaseerde aanpak het mogelijk maakt om dit te bereiken ondanks de vloek van dimensionaliteit. Hiervoor hebben we het probleem gekaderd als een KL- regelprobleem waarbij het optimal-cost-to-go een verwachte waarde zijn en dus geschat kunnen worden door middel van sampling. Op basis van de formele oplossing voor het optimale regelprobleem hebben we een geparametriseerde importancesampler ontwikkeld die de functionele vorm heeft van de optimale aansturing. We hebben een adaptive importance-sampling algoritme voorgesteld dat deze parameters zodanig aanpast dat de importance sampler de optimale aansturing benadert, waardoor de sampler opimaal woord. Op deze manier konden we de optimale kostenberekening van het KL-regelprobleem eciënt berekenen. We hebben de eectiviteit van onze methode geïllustreerd om de groei van discussieascades in een internetforum te beïnvloeden en hebben aangetoond dat deze procedure leidt tot een goede aansturingsstrategie, zelfs als een aantal van de aannames die het KL-regelprobleem deniëren, worden geschonden.

In hoofdstuk 5 hebben we een afvlakkingstechniek geïntroduceerd en bestudeerd voor de kostenfunctie van Path Integral-regelproblemen die de globale optimum invariant laat. Een gradiëntdaling op deze afgevlakte kostenfunctie denieert een spectrum van algoritmen die hebzuchtige kostenoptimalisatie bevatten voor nul
afvlakking en de onlangs geïntroduceerde PICE-methode voor oneindig sterke afvlakking. In tegenstelling tot PICE kan de gradiënt van de afgevlakte kostenfunctie echter eciënt worden geschat op basis van steekproeven als de afvlaksterkte niet te groot is. We hebben een natuurlijke gradiëntdaling overwogen en een stelling geformuleerd die het mogelijk maakt het eect van het afvlakken op het proces voor kostenoptimalisatie te begrijpen en hebben een argument gegeven waarom afvlakking de optimalisatie kan versnellen. We hebben deze theoretische resultaten met succes getest met behulp van numerieke experimenten die aantonen dat een natuurlijke gradiëntdaling op de afgevlakte kostenfunctie superieur is aan hebzuchtige optimalisatie van de niet-afgevlakte kosten en, in tegenstelling tot PICE, nog steeds haalbaar is.
Curriculum Vitae

Dominik Thalmeier was born on August 17, 1985 in Wasserburg am Inn, Germany. He grew up in Dorfen, where he finished grammar school in 2002. After one year of civil service he started studying physics at the university of Munich. During that time he spent half a year in Nijmegen for a study exchange. The time there sparked his interests in theoretical neuroscience and machine learning. After he obtained a MSc. degree in physics he moved to the Netherlands and became a PhD student in the group of Bert Kappen at the Radboud university Nijmegen to work on optimal control and neural networks.
List of publications


Donders Graduate School for Cognitive Neuroscience

For a successful research Institute, it is vital to train the next generation of young scientists. To achieve this goal, the Donders Institute for Brain, Cognition and Behaviour established the Donders Graduate School for Cognitive Neuroscience (DGCN), which was officially recognised as a national graduate school in 2009. The Graduate School covers training at both Master's and PhD level and provides an excellent educational context fully aligned with the research programme of the Donders Institute.

The school successfully attracts highly talented national and international students in biology, physics, psycholinguistics, psychology, behavioral science, medicine and related disciplines. Selective admission and assessment centers guarantee the enrolment of the best and most motivated students.

The DGCN tracks the career of PhD graduates carefully. More than 50% of PhD alumni show a continuation in academia with postdoc positions at top institutes worldwide, e.g. Stanford University, University of Oxford, University of Cambridge, UCL London, MPI Leipzig, Hanyang University in South Korea, NTNU Norway, University of Illinois, North Western University, Northeastern University in Boston, ETH Zürich, University of Vienna etc. Positions outside academia spread among the following sectors:

- specialists in a medical environment, mainly in genetics, geriatrics, psychiatry and neurology,
- specialists in a psychological environment, e.g. as specialist in neuropsychology, psychological diagnostics or therapy,
- higher education as coordinators or lecturers.

A smaller percentage enters business as research consultants, analysts or head of research and development. Fewer graduates stay in a research environment as lab coordinators, technical support or policy advisors. Upcoming possibilities are positions in the IT sector and management position in pharmaceutical industry. In general, the PhDs graduates almost invariably continue with high-quality positions that play an important role in our knowledge economy. For more information on the DGCN as well as past and upcoming defenses please visit:

http://www.ru.nl/donders/graduate-school/phd/