Neurally inspired models of belief-propagation in arbitrary graphical models

Author(s):
Steimer, Andreas

Publication Date:
2012

Permanent Link:
https://doi.org/10.3929/ethz-a-007623280

Rights / License:
In Copyright - Non-Commercial Use Permitted
Neurally Inspired Models of Belief-Propagation in Arbitrary Graphical Models

A dissertation submitted to the

ETH ZÜRICH

for the degree of

Doctor of Sciences

presented by

Andreas Steimer

Dipl.-Ing. Mikrosystemtechnik,
Albert-Ludwigs Universität Freiburg im Breisgau

born January, 21th, 1978

citizen of Germany

accepted on the recommendation of

Prof. Dr. Rodney J. Douglas, examiner

Dr. Shih-Chii Liu, co-examiner

Prof. Dr.phil.nat. Walter M. Senn, co-examiner

2012
Abstract

In recent years psychophysical experiments have revealed information processing in the human brain to be largely consistent with Bayesian inference in graphical models. Likewise, in an attempt to unify hitherto existing approaches to learning problems, parallel ongoing research in the field of machine learning has converged to the same framework. A natural question then to ask is, what is the physical nexus between the abstract Bayesian techniques from machine learning and information processing in the brain, or, more precisely, how can the known biophysical properties of neurons and their networks give rise to such Bayesian computations?

By focussing particularly on the Belief-Propagation (BP) algorithm, this thesis provides answers to that question. Our answers reside on different levels of abstraction, ranging from abstract, spike-based principles that deal with the problem at the level of neural coding, to more concrete approaches for implementing BP in neural substrates.

In the first case, we define a general sampling-based scheme that entails BP as a statistical meta-property, such that no special-purpose processing blocks for emulating the elementary BP-computations are needed. Using an interspike-interval and a spike-count code respectively, two BP processors are deduced from this scheme and verified by simulation. In particular we show that the interspike-interval processor is extremely general, since it allows for BP in arbitrary graphical models, even in presence of analog variables.

Concerning the concrete BP implementations we describe and simulate a setup of interconnected Liquid-State Machines (LSMs) as an ensemble of locally interacting nodes that give rise to BP. This general message-passing architecture was inspired by the stereotypical, graph-like connectivity observed in a variety of mammalian neocortices. Moreover, the architecture is consistent with - and provides a reasonable interpretation for- the hypothesized, canonical microcircuitry in cortex.

Finally we combine the above two levels of abstraction, by concretely implementing the sampling-based processors in our LSM setup. With respect to the interspike-interval processor we furthermore provide an interpretation of its functionality in terms of single-neuron processing. In all cases we found our simulation results to be in very good agreement with numerical evaluations of the BP algorithm.

In summary this thesis pushes forward flexible, spike-based models of BP, that reside on several levels of abstraction. At each such level, the respective models are general enough to give rise to a multitude of more concrete versions that can be experimentally tested, and of which we verify an exemplary selection by simulation. Therefore, the validity of our general models remains, even should subsequent experimental research specifically falsify their concrete versions. Moreover, some of these models provide an algorithmic meaning to the long speculated function of cortical microcircuitry, whereas others allow for BP in analog...
graphical models - a setup found to be intractable in traditional software versions of BP. Hence, the latter models may also serve as fruitful inspiration for unconventional hardware implementations, e.g. in the context of 'analog Very-Large-Scale-Integrated' electronic circuits.
Zusammenfassung

In den letzten Jahren konnte durch psychophysikalische Studien gezeigt werden, dass das Resultat der Informationsverarbeitung im menschlichen Gehirn in weiten Teilen mit bayesischer Inferenz in graphischen Modellen übereinstimmt. Im Versuch bis anhin existierende Lösungsansätze für Lernprobleme zu vereinheitlichen, griff auch die Forschung im Bereich des maschinellen Lernens parallel dazu immer stärker auf bayes'scher Methoden zurück. Es lässt sich daher die Existenz eines physikalischen Nexus' zwischen den abstrakten, bayes'schen Techniken, wie sie aus dem maschinellen Lernen bekannt sind, und der Informationsverarbeitung im Gehirn vermuten. Genauer gesagt stellt sich die Frage, wie einzelne Neurone bzw. neuronale Netzwerke es schaffen, mit ihren biophysikalischen Eigenschaften bayes'sche Berechnungen durchzuführen.

Indem sie sich speziell auf den 'Belief-Propagation' (BP)-Algorithmus konzentriert gibt die vorliegende Arbeit Antworten auf diese Frage, wobei sie von verschiedenen Abstraktionsebenen Gebrauch macht, welche von allgemeinen, spikebasierten Prinzipien bis hin zu konkreten Implementierungen des BP-Algorithmus' in neuronalen Substraten reichen.

Im abstrakten Fall, d.h. wenn das Problem auf der Ebene der neuronalen Kodierung behandelt wird, zeigen wir ein allgemeines Verfahren auf, aus welchem die BP-Berechnungsvorschriften als statistische Meta-Eigenschaft hervorgehen. Wir leiten zwei Arten von BP-Prozessoren aus diesem Verfahren ab und verifizieren diese durch computergestützte Simulation. Die beiden Prozessoren unterscheiden sich in der Art der Spikekodierung der für den Algorithmus wichtigen Informationen: Im ersten Ansatz werden Informationen durch das Zeitintervall zwischen zwei Spikes (Interspike-Intervall) übertragen, im Zweiten dagegen durch die Anzahl der Spikes. Wir zeigen auf, dass insbesondere der Interspike-Intervall basierte Prozessor sehr allgemein einsetzbar ist, da er BP in beliebigen graphischen Modellen, auch solchen mit analogen Variablen, ermöglicht.

Im Fall konkreter BP-Implementierungen in neuronalen Netzwerken hingegen beschreiben und simulieren wir eine Architektur von miteinander verknüpften 'Liquid-State Machines' (LSMs), welche die vom BP-Algorithmus benutzten, lokal interagierenden Knoten eines graphischen Modells imitieren. Diese Architektur wurde durch die stereotype, graphenartige Verbindungsstruktur motiviert, wie sie zwischen Neuronen im Neokortex in einer Vielzahl von Säugetieren vorkommt. Darüberhinaus befindet sich dieser Ansatz in Übereinstimmung mit der Idee kortikaler, kanonischer Mikroschaltkreise und weist diesen algorithmische Bedeutung zu.

Zuletzt kombinieren wir die beiden genannten Abstraktionsebenen, indem wir die abstrakten Ansätze mithilfe der LSM-Architektur implementieren. Im Fall des Interspike-Intervall Prozessors ist es darüberhinaus auch möglich, dessen Funktionalität durch die Informationsverarbeitung eines einzelnen Neurons zu
Acknowledgements

First and foremost, Prof. Dr. Rodney Douglas deserves my special thanks for offering a very open and creative working environment. He is one of the rare persons in science who first looks at the general potential of new approaches, rather than to shoot them down based on detailed technical arguments. Apart from that, he has always been a very generous and friendly supervisor, who never refused to help, even if that help involved a lot of hard work. I am well aware that having a supervisor with such qualities cannot be taken for granted. Thank you for all of that!

I like to thank Dr. Shih-Chii Liu for her careful evaluation of my thesis manuscript. Moreover, without her extensive administrative support, my doctoral defense would possibly still be in limbo, since we had almost not found a second co-examiner shortly before my defense date. Thank you so much for your support! You definitely spared me some sleepless nights by bringing Prof. Dr. Walter Senn on board who readily agreed to be my second co-examiner, despite his tight schedule on the defense date and the traveling difficulties this had caused for him. Thank you also Walter very much!

Without Florian Blättler, I would miss many hours of laughing during our usual coffee breaks (often to the dislike of Rodney, sorry for that). Thank you, Florian, for being a friend and comrade since my earliest time at the Institute of Neuroinformatics and for your corrections of the proofs in section A.1. In the same context I would also like to thank Dr. Michael Pfeiffer for the accuracy with which he corrected my paper manuscripts. Like Rodney, you have never refused to help, even if that help involved a lot of work.

Finally I want to express my gratitude to my parents Marianne and Fritz and my girlfriend Annica. Without your devotion and emotional support, this thesis would have hardly been possible. During the ups and downs of a PhD, you had always a sensitive flair for the situation at hand and created a warm and pleasant atmosphere, which always rebuilt my motivation from anew.

Thanks for all your love!
## Contents

1 Introduction .......................................................... 5  
   1.1 General Description of Thesis Objectives, Thesis Outline .................. 5  
   1.2 Motivation: Experimental Evidence for Probabilistic Computations in Nervous Systems and Parallel Developments in Machine Learning ....................... 6  
   1.3 Precise Definition of Thesis Objectives ........................................... 10  

2 Background ........................................................... 11  
   2.1 Probabilistic Graphical Models .................................................. 11  
   2.2 Biologically Plausible Inference in Graphical Models: The Belief-Propagation Algorithm ................................................................. 16  
   2.3 Relevant Properties of Spiking Neuron Models for Neural Implementations of the Belief-Propagation Algorithm .............................................. 24  
   2.3.1 The Leaky-Integrate-And-Fire Model Neuron ................................... 24  
   2.3.2 Current-Based Synapses .......................................................... 26  
   2.3.3 Escape Noise Models and Basics of Renewal Theory .......................... 27  
   2.3.4 The Diffusive Noise Model ...................................................... 27  
   2.4 Relevant Mechanisms for Learning in Spiking Neural Networks in the Context of Belief-Propagation: Reservoir Computing ........................................... 29  

3 Results I: Abstract Functional Models of Spiking Processors Implementing the Belief-Propagation Algorithm ........................................... 33  
   3.1 A Meta-Principle for Event-Based Implementations of the Belief-Propagation Algorithm ................................................................. 34  
   3.2 An Abstract Processor Based on Interspike-Intervals for Belief-Propagation in Analog Graphical Models ...................................................... 37  
   3.2.1 Functional Description of the Processor ......................................... 37  
   3.2.2 Simulation Results of Non-Neural Implementations of the Abstract Processor ................................................................. 41  
   3.2.2.1 Computation of BP-Messages in a Single Factor Node .................. 41
## CONTENTS

### 3.2.2.2 Belief-Propagation in Full FFGs

3.2.2.3 Computation of BP Messages With Biologically Realistic ISI Distributions

3.2.2.4 Inference Over Time

### 3.2.3 Implications of the Abstract Processor For Learning in Analog Graphical Models Using Expectation-Maximization

3.2.3.1 Nonparametric EM

3.2.3.2 Parametric EM

### 3.3 Functional Description of an Abstract Processor Performing Spike-Based Belief-Propagation in Discrete Graphical Models

3.3.1 Examples Of Spike-Based Events

### 3.4 Chapter Summary & Discussion

### 4 Results II: Implementing Belief-Propagation in Concrete Spiking Neural Networks Based on Liquid-State Machines

4.1 A General Model for Implementations of Message-Passing Algorithms in Neural Populations

4.2 Encoding Belief-Propagation Messages with Population Rate Codes

4.2.1 Message Update Equation, Population Rate Coding And Training of The Liquid-State Machines

4.2.1.1 Message Update Equation

4.2.1.2 Population Rate Coding of BP Messages

4.2.1.3 Setup And Training of The Liquid-State Machines

4.2.2 Computation of Single Messages

4.2.3 Inference in Whole Graphs

4.2.3.1 A Coding Problem: Inferring Signals From Unreliable Channels

4.2.3.2 Modeling a Biologically Inspired 'Explaining Away' Phenomenon

4.3 Chapter Summary & Discussion

### 5 Results III: Casting Abstract Models For Spike-Based Belief-Propagation Into Networks of Neural Elements

5.1 Concrete Neural Architecture for the Abstract Belief-Propagation Processor of Section 3.2

5.1.1 Functional Description of the Architecture

5.1.2 Performance Analysis

5.2 Concrete Neural Implementation of the Abstract Belief-Propagation Processor of Section 3.3

5.2.1 Simulation of a 2-Way Factor Node
Chapter 1

Introduction

1.1 General Description of Thesis Objectives, Thesis Outline

What fundamental problems do biological agents face during interaction with their environment? Suppose an agent has learned a sensible model of how certain events—say an approaching tiger—and objects—say an orange-black striped pattern—relate to each other. Upon observing such a pattern and given the many possible causes of striped patterns, it is vital for the agent to figure out the probability of a tiger’s fur being present and to execute appropriate behavioral responses. Assessments like this are subsumed by the term Probabilistic or Bayesian Inference and during the past two decades, a substantial number of psychophysical studies has provided strong evidence for such methods being at the core of information processing in nervous systems. Likewise, ongoing parallel research in the machine learning domain has revealed bayesian inference to be a unifying framework for a vast number of traditional approaches to learning problems.

This thesis is based on the assumption that this exciting convergence of two mostly independent fields is not coincidental. Subject to survival pressure, we believe nervous systems to have evolved general and efficient procedures for bayesian inference. The Belief-Propagation (BP) algorithm can serve as prime example of the latter, which immediately raises the question of how it can be performed by the known properties of nerve cells (neurons) in the neocortex. Providing answers to that question is the main objective of this thesis. Our answers reside on different levels of generality, ranging from abstract approaches that are not committed to specific biophysics of neurons, to more concrete neural implementations of inference processors. In most cases the latter will be concrete versions of the former. Hereby we push forward versatile, abstract approaches, whose respective validity remains, even should subsequent experimental research falsify their specific implementations.

Before we define the precise objectives of this thesis (section 1.3), we consequently lay out in more
detail the neuroscientific, experimental research that has triggered our assumption, alongside with parallel developments in the field of machine learning (section 1.2). It will become clear that BP is a sensible candidate for biologically plausible inference in neural systems. Not only allows this algorithm to perform the psychophysically relevant types of inference, it also fits elegantly to results from neuroanatomy and is based on parallelized, local information processing. Thereby it may also provide meaning to the hypothesized existence of cortical microcircuits.

Chapter 2 then provides the background material that is necessary for a complete understanding of the thesis’ results. These results are presented by starting from the most abstract level (chapter 3), continuing with a more concrete strategy for neural implementations of BP (chapter 4) and ending with combinations of the abstract and concrete approaches (chapter 5). In the discussion & outlook part (chapter 6) we summarize our results and describe in what way they can trigger future research. Specific comparisons between our neural inference models and preexisting ones are done at the end of each ‘Results’ chapter. There we also describe their connections to findings from experimental neurophysiology.

But before that, let us first review the exciting developments in the fields of psychophysics, machine learning and neuroanatomy, which all seem to point to a common underlying scheme: Bayesian Inference

1.2 Motivation: Experimental Evidence for Probabilistic Computations in Nervous Systems and Parallel Developments in Machine Learning

Prior to defining the precise goals of this thesis, we shortly review some key developments in the fields of psychophysics, machine learning and neuroanatomy. This way the convergence towards bayesian methods in the first two cases will become clear to the reader which, together with corresponding results from neuroanatomy, has motivated this thesis’ main objective of implementing bayesian computations using spike-based principles.

Behaving biological agents face a variety of information processing problems imposed by the external environment. A sensible tool for solving these problems is an internal representation of the dependencies between relevant environmental and the agents’ sensory variables. More concretely, if an agent’s nervous system contains a model that mirrors the factual relations between these variables, the model may be used as a starting point for deducing \(\text{inferring}\) environmental states, of which the agent only has indirect knowledge via its sensors. In a classical cue combination task for example \cite{38} the agent may have to estimate the true height of some object (environmental state) from visual and haptic information (sensory states). Knowledge of how the visual and haptic sensors depend on the objects height may be combined
with the measured states of both sensors, to obtain a close-to-optimal estimate (that is an estimate with the least possible variance [38]). Any representation of such variable dependencies inside the agents nervous system will be referred to as *internal model* in the following.

For example, internal models can be thought as graphs, where e.g. the nodes correspond to variables and the links to dependencies between variables (indeed, we will later identify internal models with *probabilistic graphical models*, see section 2.1). The notion of internal models is quite general and a wealth of psychophysical results has revealed that in many cases the behavior of biological agents is consistent with bayesian inference, in particular with estimation based on probabilistic graphical models. This is true for cue combination problems [38, 53, 4, 11] and tasks involving fusion of prior information and sensory evidence [105, 106, 42, 175, 77]. Unlike cue combination, estimation based on prior integration amounts to weighing sensory signals against knowledge the agent had *prior* to sensory experience, rather than weighing different sensory modalities against each other. Moreover, bayesian inference can model the well known ability of the mammalian brain to perform invariant object recognition under a variety of object transformations. For example, the pattern of light caused by some object and impinging on an observers retina may vary drastically depending on the objects position, rotation and size. Nevertheless, as it is well known from everyday life experience, the brain can easily recognize the same object irrespective of the states of these variables. Special purpose neural models have been formulated that specifically aim at achieving such invariant object recognition [48, 117, 148, 139, 160, 159], the much more general framework of bayesian inference however entails this property as a special case through the discounting of confounding variables [71, 70].

Probabilistic graphical models also allow for the modeling of dependencies between *external* variables. This way they become a model of the world and computation in such a model may be viewed as a form of proto-cognitive processing. The phenomenon of *explaining away* for example involves changing beliefs in the states of external variables of the world depending on changes in the beliefs of other such variables [125, 177, 147, 70, 74]. In particular, psychophysical results showing how object shape can influence interpretations of material properties (reflectance, color) [74, 16] have been accounted for by this framework [70]. Similarly, perceptual grouping of moving stimuli has been reported to depend on separate auxiliary stimuli [92, 113]. In case of [113] perception could even be related to cortical activity, as assessed by functional MRI. Later in this thesis we will show in greater detail how explaining away has been used for describing the perceptual illusion of [74] and present a spiking neural network that can perform inference in a corresponding graphical model.

In total the above results strongly suggest that, at least in the human brain, information processing can be captured by the framework of bayesian inference in graphical models. Interestingly, parallel, ongoing research in the field of machine learning has revealed the output of many traditional learning
machines (like perceptrons, artificial neural networks and clustering algorithms) to correspond to bayesian inference. For example the well known \textit{K-means} clustering algorithm can be generalized through a reformulation in terms of probabilistic graphical models, leading to probabilistic estimates of cluster memberships, rather than discrete, fixed assignments. Additionally the resulting procedure provides an estimate of the size and shape of each cluster region in the input space. This way bayesian methods can also be used to solve classification problems in a more general way compared to standard classifiers, which produce rather coarse class regions (e.g. simple discrimination lines in case of the perceptron). The same holds true for regression problems, where such methods allow for a representation of a full joint probability distribution, which clearly is a more general model than a parameterized functional relationship between variables. In linear regression for example data-based estimations of variable dependencies are restricted to linear functions and finding the best-fitting line can be interpreted as finding the direction of largest variance in the data. This direction however is only one specific aspect of the data, whose full joint distribution, once again, may be expressed by probabilistic graphical models.

Graphical models are not only a natural means of representing variable dependencies, they have also given rise to efficient algorithms for bayesian inference. In particular, the common theme behind inference in the above cases - in machine learning as well as psychophysics - is the estimation of a posteriori state-probabilities for variables whose state is unknown (hidden), based on variables that are in distinct observed states. The Belief-Propagation (BP) algorithm serves this goal and in section 2.2 we will deal with it in greater detail. Here we just note that BPs efficiency is due to the local and parallelized computation it employs. The nodes of the graph individually process information that is exchanged only between the nodes’ respective neighbors (message-passing). Such message-passing schemes are biologically plausible, since they lack any global observer. Therefore they have also been used in functional models of cortical processing.

So far we have seen that there has been a convergence of two largely independent fields (psychophysics and machine learning) towards bayesian methods in general and probabilistic graphical models in particular. Therefore, a natural question to ask is whether there exists neurophysiological/neuroanatomical evidence supporting the hypothesis that mammalian brains perform bayesian computations on graphical models. Although such evidence is by far not as clear-cut as in the fields of psychophysics and machine learning, we will see in the following that modern neuroanatomy has revealed interesting structures in mammalian neocortices which, as the main working hypothesis behind this thesis, can be interpreted as physical realizations of graph-based processing.

When neuronal tracers are injected into the superficial layers of the cortex and are allowed to be processed for several days, a nonuniform, patchy distribution of the tracer material develops.
Neuronal tracers are applied substances that are transported inside the axon, either away from the soma that gives rise to the axon (anterograde tracing), or in the reverse direction (retrograde tracing), or both (bidirectional tracing). Histochemical techniques can then be used to mark the paths the tracers have passed along, thereby uncovering the axonal connectivity of the neurons which have been affected by the tracer. The left-hand side of figure shows the resulting tracer distribution after such a procedure. Interestingly, the shown structure is reminiscent of a graph representing variable dependencies and it is qualitatively preserved across mammalian phylogeny, ranging from tree shrew to macaque monkey to humans. The regions of high tracer concentration (patches) consist of somas and axonal arborizations, which allows for an interpretation of the setup as an arrangement of interconnected, local circuits of neurons. Following this idea the somas represent projection neurons that send out information from a local circuit to a neighboring such circuit.

The interpretation of patchy, horizontal connectivity as a physical realization of message-passing in graphical models (fig. ) forms the basis for many results of this thesis. In particular we will provide a model that may serve as local template circuit for nodes performing message-passing in a graphical model. Therefore our approach is consistent with the notion of canonical microcircuitry, i.e. with the idea of stereotypical functional processing based on stereotypical anatomical, physiological and histochemical findings across cortical areas. The approach is furthermore consistent with biology’s constraint of local information processing.

With the above relevant background material in mind, we are now ready for a precise definition of the thesis’ objectives.

Figure 1.1: Patchy, horizontal connectivity in cortex interpreted as physical realization of graph-based computations
Left: Shown is the distribution of cholera toxin B tracer (black) in cortical layers 2/3 after its injection into all six layers (black oval). The patchy clusters of tracer consist of axon terminals and labeled somata (black dots). (figure taken from Angelucci et al.)
Right: The system of patches is reminiscent of a graph of interconnected nodes. Message-passing algorithms that play out on probabilistic graphical models are a plausible working hypothesis for the computational meaning of this anatomical finding.
1.3 Precise Definition of Thesis Objectives

Motivated by the arguments given in the previous section, the general objective of this thesis is to provide implementations of the Belief-Propagation (BP) algorithm that are based on fundamental principles of neural information processing. More concretely we head for

1. Abstract functional models of how processing with spikes lends itself to implementations of the sum-product rule, the basic computational rule of BP. These models should be functional, insofar as they focus on spike coding and computation, abstracting away from concrete biophysical mechanisms. This way, flexible models of neural BP are obtained that are not tied to the often approximative mathematical descriptions of neuronal biophysics. In this context we also investigate potential algorithmic benefits of spike-based processing compared to standard software implementations of BP.

2. Concrete implementations of BP using biophysical properties of single neurons. This objective is complementary to the one in 1 since we aim at BP dynamics that play out on a substrate of simulated neural elements (somas, dendrites, axons etc.). The goal is to find biophysical properties of neurons that are analogous to the functionality of the models developed in 1, thereby allowing single neurons to be the carrier of BP message-passing.

3. Concrete embeddings of BP in networks of neurons. In contrast to 2 we here investigate how simulated interactions of neurons in an ensemble can lead to BP message-passing. For that we utilize simplified neuron models of the 'integrate-and-fire' type. The thus obtained models should provide a reasonable interpretation for the patchy horizontal connectivity in mammalian cortices (see previous section).

Apart from approximative biophysics, there is a further advantage offered by predefining functional models (objective 1) before concrete neural implementations are looked at (objectives 2 and 3). Such an approach crystallizes core properties of the computational rule one heads to emulate, properties for which multiple concrete implementations might exist and which remain valid, even should specific implementations ever be falsified by experimental research. Indeed, as we will see later in this thesis, the sum-product rule is a prime candidate for being tackled by sampling methods, for which in turn a multitude of concrete neural implementations exist. Objectives 2 and 3 then provide such implementations but, given this argument, they have mainly an exemplary character.

With the thesis’ objectives defined, it is now time to provide the necessary background material for a thorough understanding of the results presented in the subsequent chapters.
Chapter 2

Background

In this chapter, the background material relevant for the results presented in the "Results"-chapters is provided. Readers familiar with the subjects of individual subsections may omit those as needed.

First, the general framework of probabilistic graphical models -and the inference algorithms they give rise to- is reviewed and the reader may recognize their relevance for information processing in biology (see also section 1.2). In particular the Belief-Propagation algorithm is laid out in detail. Then some information about abstract spiking models of biological neurons is provided. These models of the 'integrate-and-fire' type are conceptually placed between detailed biophysical models of the Hodgkin-Huxley type [60, 68, 55] , and even more abstract models that use a nonspiking, continuous firing rate description of neuronal activity [58, 55]. The chapter ends with a short introduction to reservoir computing, a framework that permits the implementation (learning) of circuits based on abstract spiking neurons with a distinct, predetermined functionality.

2.1 Probabilistic Graphical Models

As laid out in section 1.2 having an internal model of the world is of paramount importance for biological agents. Probabilistic graphical models as described in this section are a collection of very general approaches that serve this goal by providing an explicit description of a joint probability distribution over relevant variables. This description can then be used by the agent to perform inference, i.e. for queries about unknown variables in presence of information about known ones. In the following, probabilistic graphical models will be simply referred to as graphical models or GMs for brevity. In this section, we describe three types of GMs; Bayesian Networks, Factor Graphs and Forney Factor Graphs, and also highlight relations between them. The section ends with the conclusion that the latter two types are the best suited for modeling intelligent behavior in biological agents, a point that is refined even further in
section 2.2. It needs to be emphasized however that we only provide a coarse overview over the subject of GMs here, the reader interested in the more intricate details is directed to the literature \cite{125, 90, 15, 76}.

Common to all GMs is the idea of describing dependencies between variables as dependencies between nodes in a graph structure, e.g. in case of a biological agent between variables of the external world and the agents internal variables. For example, an animal might have an internal conception (= binary variable) about the presence of a tiger, that gets "activated" (= takes on a high probability) whenever there is a simultaneous presence of "striped fur" and "loud roar" in the environment. Activation of the "tiger" variable may then affect the probabilities of executing behavioral modes like "continue foraging" or "go into hiding" (which can also be represented as binary variables). Additionally, in case of the "hiding" mode also the quality of the animals environment is important –only if the terrain is rugged there exists a possibility for hiding. In GMs such scenarios are illustrated by graphs, where, for example, the set of nodes corresponds to a set of variables and the edges to dependencies among those variables. Such illustration however not only serves the purpose of explicitly displaying dependencies, but also to represent a factorization of the joint probability of all variables. Types of GMs only differ in the way this factorization is made, which allows for their broad classification as either directed or undirected GMs. Amongst the best known type of graphical models are Bayesian Networks (BNs) \cite{125, 15, 76}, which belong to the class of directed GMs. Here each node can be interpreted as a variable and, at the same time, as a conditional probability that is part of the factorization. Types of GMs only differ in the way this factorization is made, which allows for their broad classification as either directed or undirected GMs. Amongst the best known type of graphical models are Bayesian Networks (BNs) \cite{125, 15, 76}, which belong to the class of directed GMs. Here each node can be interpreted as a variable and, at the same time, as a conditional probability that is part of the factorization. Figure 2.1 illustrates this for the above tiger example. The network represents the factorized joint probability

\[
P(F, S, T, R, E, H) = P(H|T, R) \cdot P(E|T) \cdot P(T|F, S) \cdot P(F) \cdot P(S) \cdot P(R)\]  \hspace{1cm} (2.1)

for the observed, binary sensory variables ‘F’, ‘S’ and ‘R’ (visual presence of striped fur, auditory presence of roaring sound and presence of rugged terrain respectively), the unobserved concept variable ‘T’ (presence of a tiger) and the unobserved, behavioral variables ‘E’ and ‘H’ (continue foraging/eating and go into hiding respectively). Each node in the network 2.1 represents one of these variables and, at the same time, one factor of the above product of conditional probabilities. A variable can either be observed, when its state is known with certainty, or unobserved (hidden), in which case there is a probability distribution over its states. The presence of an edge (= arrow) between two nodes illustrates a statistical dependency between the associated variables. The node at the beginning of an arrow is called the parent of the child node at the end of the arrow. The parent node corresponds to one of the conditioning variables of the probability distribution associated with the child node. In figure 2.1 for example, \(P(T|F, S)\) is associated with node \(T\), where \(T\) is a child of its parents \(F\) and \(S\), which both project an arrow to \(T\). The resulting \(P(T|F, S)\) then captures the notion that external sensory cues, like
a roaring sound or a striped pattern in the animals visual field, may trigger (cause) an internal concept like "tiger". In turn, the belief in a tigers presence directly influences the continuation of foreaging/eating behavior \((E)\). Similarly, whether or not the animal should go into hiding \((H)\) depends primarily on the presence of a tiger \((T)\), but also on how suitable the terrain is for hiding \((R)\). These dependencies are captured by the conditional probabilities \(P(E|T)\) and \(P(H|T,R)\) respectively. The degree of determinism of such statistical dependencies is given by the numerical values these conditional probabilities provide for individual combinations of variable states. For example, with binary variables, \(P(H|T,R)\) can be represented by a \(2 \times 2 \times 2\) table of probability values. Probabilities close to one or zero then reflect almost perfect determinism.

The graph structure given by the set of arrows implicitly defines a set of conditional \((\in)\)dependency assertions between subsets of the variables in a BN. Whether or not two variable sets are statistically \((\in)\)dependent given a third such set can be examined on the basis of graph-structure using the so-called \(d\)-separation rules \([76]\).

BNs are widespread in the domains of neuroscience \([33,71,70,98,36,32]\) and machine learning \([15,76]\), however there exist certain types of \((\in)\)dependencies between variable sets that cannot be captured by any BN graph structure, and for which a different type of GM representation is required, the Markov Random Field (MRF) \([76]\). For the sake of brevity we do not lay out the MRF formalism here and direct the interested reader to the literature \([72,15,76]\), but we emphasize that, compared to BNs, MRFs are also limited in their ability to express variable \((\in)\)dependencies \([76]\). Although a factorized distribution that is represented by one of the two types of GMs can be translated into the corresponding other GM type, by introducing new edges between variables in the graph \([15,76]\), in general such a procedure undesirably increases the maximum branching degree of the factors.

Factor Graphs (FGs) in contrast are the most general graphical models, which do not have the
representational restrictions of BNs and MRFs. That is, all other graphical models can be converted to the FG formulation, without the need to increase the maximum branching degree of the associated nodes (see [79] for examples). A FG consists two types of nodes, factor and variable nodes. Each factor node represents one of the factors that make up the joint probability and is connected to all variable nodes this factor depends on. Similar to MRFs but unlike BNs, these factors need not be normalized probability distributions, i.e. in a FG the joint probability of a given set of variables $x$ is given by:

$$P(x) = \frac{1}{Z} \prod_a f_a(x_a)$$

(2.2)

where $f_a(x_a)$ is the $a$th factor of the factorization, $x_a$ the subset of variables $x$ factor $f_a$ depends on and $Z$ is the partition function rendering the product of factors a probability distribution. Figure 2.2 shows a FG for the tiger example that is a direct translation of the BN in figure 2.1.

![Factor Graph of Tiger Example](image)

It is important to realize however that this FG is just the simplest instance of a multitude of possibilities that all equivalently model the tiger example (remember that in general the factor nodes don’t need to be normalized probabilities). However the simplest way of translating a BN to a FG is precisely the way it is done in figure 2.2 Since in BNs a node corresponds to both, a variable and a conditional
probability, each such node is represented twice in the FG, as a variable and a factor node. The individual
factor nodes must be connected to the exact same set of variables the conditional probability they are
representing depends on.

As we will see in the next section, the presence of two different node types in FGs renders inference
computations unnecessarily circumstantial –particularly in a biological setup. Forney Factor Graphs
(FFGs) [43, 90] are a special type of FGs that allow for an overcoming of this difficulty. In FFGs all
nodes correspond to factor nodes, whereas the variables are represented by edges connecting two nodes.
Therefore, in a FFG each variable may only be connected to at most two factor nodes. This may seem as
a strong restriction in representational power compared to ordinary FGs, however, as shown in [90] this
restriction can be overcome easily by introducing a specific type of factor node called ’equality constraint
factor’ \( f= (x_a) \): Assume such an equality constraint factor to be attached to \( n \) variables (edges), of which
an arbitrary one we refer to as ’reference variable’. By enforcing equality between all \( n \) variables, the
equality constraint factor can ’clone’ the reference variable. That is,

\[
\begin{align*}
f= (x_a) := \prod_{i=1}^{n} \delta(x_{a1} - x_{ai}) = \begin{cases} 0 & \text{if } \exists (i, j) : x_{ai} \neq x_{aj} \\ 1 & \text{otherwise} \end{cases}
\end{align*}
\]

where \( \delta() \) denotes the dirac-delta function. In other words \( f= (x_a) = 1 \) only if all variables \( x_{ai} \) take
on the same value and therefore the factorization of \( P(x) \) in equation 2.2 remains unaffected. However
\( f= (x_a) = 0 \) and hence \( P(x) = 0 \) for all other combinations of variable values. This way, the factor
\( f= (x_a) \) prevents any variable from assuming different values than the others and the reference variable is
’cloned’. With this type of factor node FFGs obtain the same representational power as ordinary FGs.

Figure 2.3 shows an example FFG modeling the tiger example. It can be directly obtained from the FG
of figure 2.2 by replacing variable nodes through edges (if these variable nodes are connected to at most
two factor nodes) or equality constraint factors (if these variable nodes are connected to more than two
factor nodes).

In summary, these easy and elegant translations of other types of graphical models to FGs/FFGs show
the great representational power of the latter. Any (in)dependency between variables expressed in a BN
or MRF can also be captured by a FG/FFG, without the need to introduce additional variables during
the translation process. This stands in marked contrast to BNs and MRFs. Hence FGs/FFGs cover a
greater range of representable joint probability distributions that may be relevant for biological agents.
Furthermore as we will see in the following section, FGs/FFGs also allow for much simpler formulations
of biologically plausible inference mechanisms than BNs/MRFs.
CHAPTER 2. BACKGROUND

Figure 2.3: Example Forney Factor Graph modeling the tiger example. Square nodes correspond to factors, edges connecting these factors to variables. The equality constraint factor is indicated by a ‘=’. The graph represents the joint probability distribution

\[ P(F, S, T, R, E, H) = \frac{1}{Z} \cdot f_e(T, T', T'') \cdot P(H|T', R) \cdot P(E|T'') \cdot P(T|F, S) \cdot P(F) \cdot P(S) \cdot P(R). \]

Due to the equality constraint factor \( f_e(T, T', T'') \) this is a different factorization of the same distribution \( P(F, S, T, R, E, H) \) in equation 2.1.

2.2 Biologically Plausible Inference in Graphical Models: The Belief-Propagation Algorithm

In the previous section we have seen how graphical models can illustrate (in)dependencies within a set of variables that are important for a biological agent interacting with its environment. Apart from illustrative, human readability however, such graphical representations also offer distinct advantages for bayesian inference. In this section we will describe a specific inference algorithm, the Belief-Propagation (BP) algorithm, which allows for a simultaneous inference of the (conditional) marginal probability distributions of all variables of interest in the GM. BP is of particular relevance in biological contexts, since its underlying computations are local, i.e. are devoid of any global observer and can be performed by processing units that communicate locally through distribution of information to their neighbors only (message-passing). Although there exist other inference procedures that are based on message-passing and which compute other biologically relevant quantities, e.g. the max-product algorithm for computing the maximum a pos-
teriori (MAP) estimate, for two reasons we here focus on BP only: Firstly, the outcome of max-product can often be extracted from the steady-state BP belief (by taking the maximizing argument of the beliefs) [173, 44, 174, 176], but not vice versa. Therefore computing the marginals seems to be a more general inference procedure. Secondly, as we will see in later in this thesis the marginals also play an important role for learning in the context of graphical models. Readers who are interested in other inferential schemes in graphical models besides BP are referred to the literature [125, 44, 3, 103, 76].

For didactic reasons in this section we first describe BP in Forney Factor Graphs and subsequently also in ordinary Factor Graphs and Bayesian Networks. Although this order corresponds to the inverse historical development, it is a surprising fact that BP in FFGs is both most simplest and most intuitive. As we will see in later sections this simplicity turns out to be of great importance for biologically plausible implementations of BP in neural systems.

As stated before, the goal of inference by means of BP is to simultaneously compute the marginal distributions of a set of unobserved variables, eventually in presence of a set of ‘evidence’ (or observed) variables whose state is known. In the tiger example of figure 2.3 a prey animal must decide between fleeing, escaping and doing nothing (unobserved variables), based on available sensory evidence about striped fur, roaring sound and rugged terrain (observed variables). Let us assume for the moment that there is no such reliable sensory evidence available to the animal, i.e. only ambiguous visual input and a cacophony of sounds –in other words none of the variables are observed. For figuring out what action to take, the animal needs to compute the unconditional marginals $P(E)$ and $P(H)$ from the joint distribution $P(F, S, T, R, E, H)$:

$$P(E) = \sum_{F,S,T,R,H} P(F, S, T, R, E, H)$$

$$= \frac{1}{Z} \cdot \sum_{F,S,T,T',T'',R,H} f_{-}(T, T', T'') \cdot P(H|T', R) \cdot P(E|T'') \cdot P(T|F, S) \cdot P(F) \cdot P(S) \cdot P(R)$$

$$P(H) = \sum_{F,S,T,R,E} P(F, S, T, R, E, H)$$

$$= \frac{1}{Z} \cdot \sum_{F,S,T,T',T'',R,E} f_{-}(T, T', T'') \cdot P(H|T', R) \cdot P(E|T'') \cdot P(T|F, S) \cdot P(F) \cdot P(S) \cdot P(R)$$

In other words, the joint probability has to be summed over the domains of all variables in the model, except $E$ and $H$ respectively. This procedure is very costly, since for each joint state of the seven variables ($\{F, S, T, T', T'', R, H\}$ and $\{F, S, T', T'', R, E\}$ respectively) a product of seven factors has
to be computed. If we assume all variables to be of the binary type this means that we have to evaluate $6 \cdot 2^7 = 768$ multiplications. Although this is still manageable in our particular example, in general the number of products to be processed by straightforward marginalization increases exponentially with the number of variables in the model. On the other hand we can reduce computational effort simply by applying the distributed law, e.g. to equation 2.4:

$$P(E) = \frac{1}{Z} \cdot \sum_{T,T',T''} f_w(T,T',T'') \cdot P(E|T'') \cdot \left( \sum_{R,H} P(H|T',R) \cdot P(R) \right) \cdot \left( \sum_{F,S} P(T|F,S) \cdot P(F) \cdot P(S) \right)$$

(2.6)

The effect of such 'clustering' of summations—which lies at the core of BP— is a considerable decrease in computational cost. In equation 2.6 we only need to evaluate $3 + 2^2 + 2 \cdot 2^2$ multiplications for each of the $2^3$ joint states of the variables $\{T, T', T''\}$, leading to a total of $2^3 \cdot (3 + 2^2 + 2 \cdot 2^2) = 120$ multiplications only. The intuitive reason for this cost reduction compared to straightforward marginalization (equation 2.4) lies in the fact that a given product in 2.6 needs not be evaluated for each joint state of those variables on which the product does not depend. For example, $f_w(T, T', T'') \cdot P(E|T'')$ is only evaluated for each $(t, t', t'') \in \{0, 1\}^3$ rather than for each $(f, s, t, t', t'', r, h) \in \{0, 1\}^7$, because computing this product is redundant in $2^4$ cases.

To understand how BP simplifies marginalization we apply the distributed law a second time, yielding

$$P(E) = \frac{1}{Z} \cdot \sum_{T''} P(E|T'') \cdot \left( \sum_{T,T'} f_w(T,T',T'') \cdot \left( \sum_{R,H} P(H|T',R) \cdot P(R) \right) \right) \cdot \left( \sum_{F,S} P(T|F,S) \cdot P(F) \cdot P(S) \right)$$

(2.7)

It is important to see how these successive algebraic manipulations can be represented in the graph of figure 2.3. If we define the above terms in brackets as 'messages' $m_{i \rightarrow j}(X_{ij})$ sent from some factor node $i$ to a neighboring factor node $j$ along the connecting variable edge $X_{ij}$, we can reformulate equation 2.7 in terms of these locally computable quantities. Such messages are always functions of the variable $X_{ij}$.
connecting the two communicating factor nodes \(i\) and \(j\).

\[
m_{p_H \to f_E}(T') := \sum_{R,H} P(H|T',R) \cdot P(R)
\]

\[
m_{p_T \to f_E}(T) := \sum_{F,S} P(T|F,S) \cdot P(F) \cdot P(S)
\]

\[
m_{f_E \to p_E}(T'') := \sum_{T,T'} f_E(T,T',T'') \cdot m_{p_T \to f_E}(T) \cdot m_{p_H \to f_E}(T')
\]

Therefore,

\[
P(E) = \frac{1}{Z} \cdot \sum_{T''} P(E|T'') \cdot m_{f_E \to p_E}(T'') = \frac{\sum_{T''} P(E|T'') \cdot m_{f_E \to p_E}(T'')}{\sum_{T''} \sum_{E} P(E|T'') \cdot m_{f_E \to p_E}(T'')} \quad (2.8)
\]

Figure 2.4 illustrates how these messages relate to the structure of the FFG modeling the tiger example. The two messages \(m_{p_H \to f_E}(T')\) and \(m_{p_T \to f_E}(T)\) are computed independently and locally by the nodes \(P(H|T',R)\) and \(P(T|F,S)\) respectively. They are both sent to node \(f_E\) which subsequently computes message \(m_{f_E \to p_E}\) and sends it to node \(P(E|T'')\). Based on equation 2.8, this then allows node \(P(E|T'')\) to compute the final marginal \(P(E)\), using exclusively local information (the definition of \(P(E|T'')\) itself and incoming message \(m_{f_E \to p_E}(T'')\)). The partition function \(Z\) needs not be computed explicitly, since equation 2.8 is invariant to scalings of any of the involved messages.

The distributed nature of a message-passing scheme like this allows marginalization to be performed by an arrangement of locally communicating agents (nodes), that are devoid of any global information about all the messages in the graph. In contrast to direct marginalization by a global observer (equation 2.4), each message sent by some node only contains summations across those variables (dashed colored boxes in figure 2.4) that belong to the subgraph 'behind' the node (with respect to the direction of the message). A message can thus be seen as a summary of the subgraph behind its sending node \([90, 103]\).

For example \(m_{p_T \to f_E}(T)\) is sent from \(P(T|F,S)\) to \(f_E\) and therefore only contains summations across \(F\) and \(S\), both of which are the only variables (edges) in the subgraph behind \(P(T|F,S)\) (red dashed box). Similarly \(m_{p_H \to f_E}(T')\) summarizes over \(R\) and \(H\) (green dashed box) and \(m_{f_E \to p_E}\) over \(T\) and \(T'\). Since \(m_{f_E \to p_E}\) depends on \(m_{p_T \to f_E}(T)\) and \(m_{p_H \to f_E}(T')\), it can be regarded as a summary over \(\{F,S,T,T',R,H\}\) (blue dashed box). This 'box within boxes' principle \([90]\) is a general theme behind message-passing algorithms \([3, 103]\) and can be applied to other tasks besides marginalization, e.g. for computing the MAP estimate. The principle also makes understandable that BP yields exact results in cycle-free graphs only, since in case of cycles one cannot define proper subgraph 'boxes', meaning that
certain summations are double counted. Furthermore, it becomes clear that the BP messages can be used not only for computing single variable marginals, but also for 'joint' marginals of all those variables which are connected to the same factor node. In fig. 2.4 for example, \( P(T, T', T'') \) could be assessed through
\[
P(T, T', T'') = \int f_e(T, T', T'') \cdot m_{p_T \rightarrow f_e}(T) \cdot m_{p_H \rightarrow f_e}(T') \cdot m_{p_E \rightarrow f_e}(T'').
\]
These 'factor marginals' do not play a prominent role in this thesis, we will see in section 3.2.3 however they are crucial for learning the factor functions from data.

We will now define the general rule for computing BP output messages for factor nodes in FFGs. To be compatible with material presented in later sections we assume all variables to be analog. The discrete case however can still be assessed by replacing the various integrals in the general rule by corresponding sums. Let \( D \) be the domain of each variable \( X_k \). In addition to the single index \( k \), in FFGs we further introduce a double index \( i, j \) for the variable(edge) \( X_{ij} \) connecting factor nodes \( i \) and \( j \). For simplicity we assume all variables to have the same domain \( D \). Denote by \( m_{i \rightarrow j} \) the message sent by node \( i \) to its neighboring node \( j \in N(i) \), where \( N(i) \) is the index set of neighboring nodes of \( i \), \( n_i := |N(i)| \). Although, as indicated above, the final marginal probabilities are invariant to scalings of the individual messages,
normalization is typically needed to avoid diverging or diminishing message values during the message-passing dynamics [90]. Therefore \( m_{i \rightarrow j} \) refers to a message normalized to 1 and \( u m_{i \rightarrow j} \) to its unnormalized counterpart. \( N_{i \rightarrow j} \) is the corresponding normalization constant. Define \( N V (i) := \{ j \mid j \in N(i) \} \) to be the index set of variables(edges) attached to factor node \( i \). Finally let \( x_\mathcal{I} := (x_i \mid i \in \mathcal{I}) \) be the vector of values of variables whose indices are in the index set \( \mathcal{I} \) and \( dx_\mathcal{I} := \prod_{i \in \mathcal{I}} dx_i \) the corresponding differential. In FFGs, output messages of some factor node \( i \) defined by function \( f_i (x_{NV(i)}) \) are then computed from incoming messages according to the Sum-Product Rule (SPR) [51, 90, 91, 15, 76]:

\[
\begin{align*}
  um_{i \rightarrow j} (x_{ij}) & := \int_{D_{ni \rightarrow j}} f_i (x_{NV(i)}) \prod_{k \in N(i) \setminus \{ j \}} m_{k \rightarrow i} (x_{ik}) \, dx_{x_{NV(i)} \setminus \{ j \}} \\
  N_{i \rightarrow j} & := \int_{D} um_{i \rightarrow j} (x_{ij}) \, dx_{ij} \\
  m_{i \rightarrow j} (x_{ij}) & := \frac{um_{i \rightarrow j} (x_{ij})}{N_{i \rightarrow j}}
\end{align*}
\] (2.9)

Based on such messages an estimate \( B(x_{ij}) \) of the marginal probability \( P(x_{ij}) \) can be formed as soon as at least one message has been passed in both directions along edge \( X_{ij} \):

\[
B(x_{ij}) := \frac{m_{i \rightarrow j} (x_{ij}) m_{j \rightarrow i} (x_{ij})}{\int_{D} m_{i \rightarrow j} (x_{ij}) m_{j \rightarrow i} (x_{ij}) \, dx_{ij}}
\] (2.12)

\( B(x_{ij}) \) is typically referred to as the belief of variable \( X_{ij} \). In FFGs the beliefs need not be computed by a separate set of equations, following equation 2.11 they can rather be obtained as output messages from an equality constrained factor that is interlaced between each pair of connected factor nodes [91]. For tree structured FFGs (and other types of GMs as well) it is always the case that \( B(x_{ij}) = P(x_{ij}) \) [90, 180]. However in graphs containing cycles the \( B \)'s are only approximations to the true marginals [90, 180]. The quality of these approximations depends on the length of the cycles and on the specific instantiations of the factor nodes in the graph, often however they are surprisingly exact [180]. Also, convergence of the message update process defined by eqs 2.9-2.11 is not guaranteed in cyclic graphs. Then the message update schedule that determines, for each message, when it has to be computed and send becomes important for the algorithm to converge. The simplest such schedule, which will be exclusively used throughout this thesis, is the 'flooding schedule' [78, 79]. There, every message in the graph is computed and transmitted at every time step. Therefore, in this case processors that emulate BP do not need additional machinery for determining when a message update has to take place. Additionally, the flooding schedule can be formulated directly as a first-order time-continuous dynamical system, which can be conveniently implemented by neural circuitry (see section 4.2).

In FFGs it is apparent that besides normalization the SPR only consists of a single stereotyped update
rule (equation 2.9). As we will see below, this is very different from other graphical models, e.g. ordinary FGs and BNs. Although this aspect is not of utter importance in the machine learning domain, for two reasons it is crucial for the objectives of this thesis. First, each additional update equation possibly requires a distinct neural machinery for its computation. Hence using multiple such equations is a more complicated implementation strategy that is far more error prone. Second, with reference to Occams Razor [103], a single update rule is the sparsest possible assumption needed to explain canonical structures (patches) in cortex, and should hence be preferred to more complicated hypotheses. Following Popper, parsimonious theories are also easier to test and falsify [132].

I will now briefly describe the BP algorithm for FGs and BNs and contrast these versions to BP in FFGs. Unlike FFGs, in FGs there exist two distinct types of message update rules, one for messages from factor to variable nodes and the other one for messages in the reverse direction. Denote with \( i \) a factor node and with \( j \) a neighboring variable node. The two update rules then are as follows [79]:

- **Variable to factor node**: 
  \[
  u_{m_{j \rightarrow i}}(x_j) = \prod_{k \in N(i) \setminus \{i\}} m_{k \rightarrow j}(x_j)
  \]
  (2.13)

- **Factor to variable node**: 
  \[
  u_{m_{i \rightarrow j}}(x_j) = \int_{D_{N(i) \setminus \{j\}}} f_i(x_{N(i)}) \prod_{k \in N(i) \setminus \{j\}} m_{k \rightarrow i}(x_k) \, dx_{N(i) \setminus \{j\}}
  \]
  (2.14)

Normalization of these messages is done in the same way as in FFGs, i.e. by using equations 2.10 and 2.11. The beliefs \( B(x_j) \) can be computed by multiplying and normalizing all messages arriving at variable node \( j \) [180]:

\[
B(x_j) := \frac{\prod_{k \in N(i)} m_{k \rightarrow j}(x_j)}{\int_{D_{N(i)}} \prod_{k \in N(i)} m_{k \rightarrow j}(x_j) \, dx_j}
\]
(2.15)

Clearly equation (2.14) is identical to equation (2.9) but distinct from (2.13) since in the latter case multiplication with a local factor function and subsequent integration are both missing. Furthermore, computing the belief involves yet another distinct equation in FGs compared to FFGs. As we have seen in the previous section FFGs elegantly avoid the complications imposed by these two additional message update equations. Indeed, replacing each variable node with an equality constraint factor and applying equations 2.5 and 2.9 leads precisely to equation (2.13).
The presence of two types of message update equations is also a feature of BP in BNs \[125, 79\]. In contrast however, in BNs the distinction is made with respect to the direction of a message compared to the directed edge it is passed along. That is, messages from child to parent nodes are computed differently than messages from parent to child nodes. In both cases however a message is always a function of the parent node variable. Let $i$ be a child node of some parent $j$, denote by $P(k)$ be the set of parents and by $C(k)$ the set of children of some node $k$ and let $n_j := |P(j)|$, $m_i := |P(i)|$. Then the corresponding update equations for BP in BNs then are as follows \[125, 79\]:

- Parent to child node:
  \[
  u_{m_{j \rightarrow i}}(x_j) = \prod_{c \in C(j) \setminus \{i\}} m_{c \rightarrow j}(x_j) \int_{D^{n_j}} f_j(x_j \mid x_{P(j)}) \prod_{p \in P(j)} m_{p \rightarrow j}(x_p) \, dx_{P(j)} \quad (2.16)
  \]

- Child to parent node:
  \[
  u_{m_{i \rightarrow j}}(x_j) = \int_{D} \prod_{c \in C(i)} m_{c \rightarrow i}(x_i) \int_{D^{m_i-1}} f_i(x_i \mid x_{P(i)}) \prod_{p \in P(i) \setminus \{j\}} m_{p \rightarrow i}(x_p) \, dx_{P(i) \setminus \{j\}} \, dx_i \quad (2.17)
  \]

the belief of some variable $x_j$ can be computed in the same way as parent-to-child-messages, in contrast however all of the messages received by $j$’s children need to be incorporated:

\[
B(x_j) := \prod_{c \in C(j)} m_{c \rightarrow j}(x_j) \int_{D^{n_j}} f_j(x_j \mid x_{P(j)}) \prod_{p \in P(j)} m_{p \rightarrow j}(x_p) \, dx_{P(j)} \quad (2.18)
\]

In summary it is justified to state that, compared to FGs and BNs, BP in FFGs leads to a great simplification of the computations required for message updating. Interestingly historical development of BP began with its most complicated formulation for BNs, as described in the influential work of Pearl \[125\]. Subsequent development by Kschischang et al. \[79\] then led to the more elegant formulation of BP in FGs, which was simplified even more by Forney \[43\] with his introduction of FFGs (which he called ’Normal Graphs’). As discussed above, the simplification to a single message update rule in FFGs is very desirable in a biological context. In fact, all models discussed in the ’Results’ section use FFGs as their representational basis.
2.3 Relevant Properties of Spiking Neuron Models for Neural Implementations of the Belief-Propagation Algorithm

In this section the simplified biophysical mechanisms of neural processing that are necessary for the results in chapters 3, 4, and 5 are provided. We will first describe the deterministic dynamics underlying the Leaky-Integrate-And-Fire (IAF) neuron model, which acts as a workhorse in the theoretical neuroscience domain and also forms the basis for many of the results of this thesis. In this context, the current-based synaptic model that is solely utilized in this thesis will also be laid out. We will then describe two possibilities for imposing stochasticity on the output of such a model neuron. This is necessary because all neural models of Belief-Propagation presented in chapters 3, 4, and 5 are based on randomized activity of neurons. In the first scenario, the IAF model is slightly altered to incorporate stochasticity directly in its dynamical equations (Escape noise model [55]). This case then allows for a convenient application of classical results from renewal theory that are subsequently explained – and heavily relied upon in section 3.2.1. Alternatively stochasticity can arise through the random arrival of synaptic input to a deterministic IAF neuron, which is the topic of the second scenario (Diffusive Noise model [55]) and needed in sections 4.2 & 5.2. In the following we assume some basic knowledge about the physiological and anatomical properties of real neurons, readers unfamiliar with the field are referred to the introductory literature [68, 75, 69, 30, 55].

2.3.1 The Leaky-Integrate-And-Fire Model Neuron

IAF-type models [157, 158, 75, 30, 55] of neural behavior are simplified, abstract versions of the more detailed, biophysical Hodgkin-Huxley (HH) type of models [60, 75, 55]. The former models have been defined to simplify theoretical analysis of the latter. Also, they capture some key properties of the HH model, for example the presence of a resistive, lipid bilayer surrounding a neuron’s cell body (soma) and a voltage threshold. In contrast to the HH model that also accounts for voltage dependent conductances (ion channels) in the bilayer, in the basic IAF model [55] abstracts the bilayer by a charge-leaking sheet of membrane that only entails capacitive and resistive properties. The total input current $I_{tot}$ flowing into the cell body is therefore conceptually divided in two parts, a capacitive current that charges up the membrane and a resistive one that leaks out of the cell (fig. 2.5). The voltage $V_m$ across the membrane that develops through such charging is constraint by a reset mechanism that becomes activated as soon as a maximum value (threshold $V_\theta$) is crossed (fig. 2.5). At this point, the neuron fires an action potential (or spike) and the voltage is set back to some predefined value called the ‘reset potential’ $V_{reset} < V_\theta$. Immediately after firing a spike, the neuron enters a refractory period $t_{ref}$, during which no further spike can be fired regardless on the input current. Following $t_{ref}$ the charge-integration process of the
membrane capacitor starts anew. In the absence of any stimulating current however, the resistive current discharges the capacitor completely, ending with $V_m$ relaxing to the resting potential $V_r$.

It is important to be aware of the fact that both, the refractory period and the voltage threshold, are mere phenomenological abstractions of the biophysical properties that emerge from the dynamics of the more detailed HH model neuron [68]. Also, in IAF models the shape of an action potential is not modeled explicitly, rather a spike is treated as a binary event (present or not present) and only its impact on postsynaptic neurons is considered. With $R$ and $C$ denoting the membrane resistance and capacitance respectively and $\dot{V}_m(t) := \frac{dV_m(t)}{dt}$, the dynamics of the basic IAF model (fig. 2.5) can be expressed mathematically by the following linear, first-order differential equation:

$$
\text{if } V_m(t) < V_\theta: \quad \tau_m \dot{V}_m(t) + V_m(t) = V_r + RI_{\text{tot}}(t) \\
\text{if } V_m(t) = V_\theta: \quad \lim_{t' \to t; t' > t} V_m(t') = V_{\text{reset}}
$$

where $\tau_m := RC$ is called the membrane time constant of the neuron that determines the overall speed of the fluctuation dynamics of the membrane.

Figure 2.5: Circuit diagram of the IAF somatic model
The total input current flowing into the neurons soma is divided into a resistive part that flows through resistor $R$ and a capacitive part that charges the capacitor $C$. The voltage $V_m$ that thus develops across the $RC$ circuit is compared to the maximally allowed value (threshold $V_\theta$). If $V_m > V_\theta$ a spike is released by the neuron and $V_m$ is reset to the reset potential $V_{\text{reset}}$. 
2.3.2 Current-Based Synapses

The total current $I_{tot}$ that reaches the soma is thought to be the sum of all contributions of presynaptic partners the considered neuron is connected to. The IAF formulation of somatic activity can be combined seamlessly with a huge variety of synapse models that determine the impact of a presynaptic spike on some postsynaptic target. In general there are two classes of such models, conductance-based and current-based models [55]. As the name indicates, in conductance-based models a presynaptic spike modulates the conductance of the postsynaptic membrane in a time dependent way, thus triggering currents into or out from the postsynaptic cell. Because it is the conductance that is modulated, the magnitude of these currents also depends on the membrane voltage at the specific position of the postsynaptic site. Current based synapses in contrast lead to stereotypical postsynaptic currents that only depend on time.

In this thesis we only consider current based synapses, because they are easier to work with in the context of trainable circuits (section 2.4). However this is not just a conceptual simplification, rather, for excitatory synapses, which are the predominate type of synapses in cortex, this assumption is a reasonable approximation to neurobiological reality [55]. In particular we focus on synapses whose contribution to $I_{tot}$ can be described by the following linear, first-order differential equation [55]:

$$I_{tot}(t) = \mathbf{w}^T \cdot \mathbf{I}(t)$$

$$\mathbf{I}(t) := (I_1(t), ..., I_n(t))^T$$

$$\tau_s \frac{d}{dt} I_k(t) = -I_k(t) + \sum_i \delta(t - t_k^i - D_k)$$  \hspace{1cm} (2.20)

where $\mathbf{w}$ is the (column) vector of synaptic weights (magnitudes of the stereotypical postsynaptic currents) of all $n$ presynaptic partners of the considered neuron. $\tau_s$ is the time constant of (leaky) synaptic integration, $t_k^i$ denotes the time of the $i$-th spike of the $k$-th presynaptic neuron and $D_k$ is the transmission delay associated with the synaptic input from $k$. The dynamics defined by equation (2.20) have an intuitive interpretation: Each presynaptic spike of neuron $k$ adds a stereotypical current trace $I_\delta(t)$ to $I_k(t)$, where $I_\delta(t)$ corresponds to Green’s function (or impulse response) of equation (2.20). In this case, Green’s function follows a simple decaying exponential function:

$$I_\delta(t) = \frac{1}{\tau_s} e^{-\frac{t}{\tau_s}}$$  \hspace{1cm} (2.21)

When equations (2.19) and (2.20) are combined, the resulting dynamics of the IAF neuron must necessarily remain deterministic. This finding is consistent with experiments on intracellularly stimulated neurons in vitro [61]. In contrast however, the activity patterns of such neurons in vivo can often be reasonably described in stochastic terms [110, 23, 153, 61, 151]. This discrepancy is resolved by assuming random
arrival of balanced excitatory and inhibitory input currents at some of the synapses, input that is sub-
sequently processed in a deterministic way by the neuron [55, 80]. Stochastic openings and closings of ion
channels in the membrane of the postsynaptic neuron [178, 55, 145, 40], as well as spontaneous release
of transmitter vesicles on the presynaptic side [41, 40], can act as additional noise source. We will now
describe how the IAF dynamics of equation 2.19 can be altered to mimic the combined effect of such
stochasticity. This then permits linking neural processing to the Interspike Interval (ISI) distribution,
a core ingredient of the model presented in section 3.2.1. Subsequently we will also lay out the more
realistic case, where stochastic behavior arises from random arrival of input spikes that is processed
deterministically (as used in sections 4.2 and 5.2).

2.3.3 Escape Noise Models and Basics of Renewal Theory

The strict thresholding condition of equation 2.19 can be relaxed by allowing the neuron to fire even
when \( V_m < V_\theta \). The probability density for a spike to happen at time \( t \) (also called escape rate or hazard)
 [55] may arguably depend on \( V_m(t) \) in a monotonically increasing way. The hazard thus essentially lumps
together the above mentioned sources of noise. Typically an exponential dependency of the form

\[
h(V_m) = h_\theta \cdot \exp\left[\frac{V_m - V_\theta}{\sigma_h}\right]
\]  

(2.22)

is used in practice [37, 136, 128, 127], but other models do also exist [11, 165, 129]. The parameter \( V_\theta \)
can be regarded as a 'soft' threshold, whose crossing by \( V_m \) from below renders the neuron more likely to fire.
The 'softness' of the threshold is controlled by \( \sigma_h \) and the value of \( h \) at \( V_\theta \) by \( h_\theta \). Suppose the neuron
has fired its last spike at time \( t_0 \) and we are interested in knowing the ISI distribution \( p(\Delta t, t_0) \), i.e. the
distribution of times \( \Delta t := t - t_0 \); \( t > t_0 \) between subsequent spikes. Obviously the shape of the curve
\( h(t) := h(V_m(t)) \) has an impact on \( p(\Delta t, t_0) \), since a low\backslash high \( h \) will lead to an earlier\backslash later subsequent
spike. A classic result from renewal connects these two quantities in the following way [28, 55]:

\[
p(\Delta t, t) = h(t) \cdot \exp\left[-\int_{t_0}^{t_0+\Delta t} h(t') \, dt'\right]
\]  

(2.23)

\[
h(t) = \frac{p(t - t_0, t)}{1 - \int_{t_0}^{t} p(t' - t_0, t') \, dt'}
\]  

(2.24)

2.3.4 The Diffusive Noise Model

Whereas escape noise models describe stochastic neural behavior on a phenomenological level, the diffusive
noise model is more specific with respect to the biophysical origins of noise. Ignoring contributions of
stochastic ion channels and vesicles, the diffusive noise model assumes the random behavior of a neuron to
be caused solely by the massive bombardment of synaptic inputs from the neurons presynaptic partners. In this context, the two basic assumptions of the diffusive noise model are:

1. Balanced amounts of fast decaying excitatory and inhibitory inputs from current-based synapses. This permits abstracting the synaptic input current as a zero-mean gaussian white noise process.

2. Deterministic IAF processing of this random input current according to eq. 2.19.

If, in addition to 1, one assumes the presence of a subset of unbalanced control synapses that do not contribute to the white noise, the total input current \( I_{\text{tot}} \) of the neuron in eq. 2.19 can be written as the sum of two contributions: A (diffusive) zero-mean white noise component \( I_{\text{noise}} \) as before and a (drifting) mean component \( I \) due to the control inputs. Although the diffusive noise model is based on a concrete biophysical cause of noise in neurons, it is also possible to express it as a more phenomenological escape-rate model [129], that we described earlier. When the control inputs provide a constant mean drive to the neuron over time, that is when the statistics of the input are stationary, it is possible to analytically derive the \( r(I) \) curve, which describes the relationship between the drift current \( I \) and the neurons resulting frequency \( r \) of repetitive spike firing [24, 5, 6, 80]:

\[
r(I) = \left( t_{\text{r}} + \tau_m \frac{\dot{\theta}}{\bar{v}_c} \int \sqrt{\pi} e^{u^2} (1 + \text{erf}(u)) \, du \right)^{-1}
\]

(2.25)

where the 'hat' operation is defined by: \( \hat{z} := \frac{C z - \tau_m I}{\sigma_I \sqrt{2 \pi} \tau_m} \) and \( \text{erf}(u) \) denotes the error function, \( \sigma_I \) the standard deviation of the white noise component \( I_{\text{noise}} \) and \( \tau' = 1\text{ms} \) is a factor to preserve units.

Although eq. 2.25 has been derived for the case of stationary inputs (constant mean current \( I \) and standard deviation \( \sigma_I \) of the noise), with increasing synaptic time constant \( \tau_s \) of the control inputs it becomes a better and better approximation also for the slow transient case (weakly time dependent \( I(t) \) and/or \( \sigma_I(t) \) [80]. This condition is best fulfilled by synapses dominated by the NMDA receptor type whose time constant is around 80ms [75]. We will make use of model 2.25 in section 5.2, where the spike output of a small population of neurons is of importance.

Conversely, using a large population of neurons renders a single spike unimportant, but allows for the transmission of fast transients of the input current through population rate coding [51, 55]. Processing of fast transients is crucial for biological systems that have to react rapidly to sudden changes in their environment. Unlike the single neuron firing rate, the population rate \( R \) is defined for a large population of \( N \) neurons as the fraction of spike-emitting neurons within a small time window:

\[
R(t) := \lim_{\Delta t \to 0} \lim_{N \to \infty} \frac{1}{\Delta t} \frac{n_{\text{act}}(t; t + \Delta t)}{N}
\]

(2.26)
where \( n_{act}(t; t + \Delta t) \) is the number of firing neurons during time window \([t, t + \Delta t]\). Diffusive noise in neuronal populations may readily be used for such a coding scheme. Due to the noise there will always be some neurons hovering just below spike threshold and which, upon stimulation with a fast transient input, will fire immediately \([54, 55, 80]\). In section 4.2 we will empirically measure the \( I(R) \) relationship for such a population, in order to construct circuits which draw inferences about in the external world on psychophysically relevant time-scales.

With the relevant facts about spiking neuron models in mind, we can now have a look at mechanisms that allow for a construction of circuits of model neurons with a distinct predefined functionality.

### 2.4 Relevant Mechanisms for Learning in Spiking Neural Networks in the Context of Belief-Propagation: Reservoir Computing

In this section, we present the basic mechanisms that are crucial for understanding the concrete neural implementations of BP in sections 4.2 and 5. In essence all of these models are based on a learning framework that can be applied to spiking as well as nonspiking neuronal units: Reservoir Computing \([66, 101, 93]\). Reservoir computing (RC) refers to a collection of methods which, based on recurrently connected networks, allow for learning a broad class of filters that map input to output time-series. In general, the basic units that form connections in these networks can belong to a broad range of nonlinear dynamical system models \([93]\), in this thesis however only units that either belong to some spiking or nonspiking neuronal model are considered. Therefore, throughout the remaining thesis these basic units will all be called 'neurons'. More specifically, the results of section 4.2 are based (spiking) IAF neurons and those of section 5.1 on (nonspiking) sigmoidal neurons \([58, 67]\) (the workhorse of Artificial Neural Network modeling). In the former case the RC network is termed a Liquid-State Machine (LSM) \([101]\), whereas the latter case is referred to as Echo-State Network (ESN) \([66]\). Throughout the thesis the term 'filter' (or 'operator') refers to mappings from input to output time series. For example, a simple current-based synapse described by equation 2.20 is a low-pass filter that maps an arbitrary spike train to the corresponding postsynaptic current.

The common goal of RC methods is to approximate (or learn) a filter \( F \), based on samples from some input time-series \( u(t) \) and the corresponding filter output \( y(t) = [F(u)](t) \) (for simplicity we restrict the discussion to a single input time series, although in general multiple inputs may be present). This goal is achieved by utilizing a common network architecture that consists of three distinct neuronal populations (figure 2.6):
1. Neurons that are part of a Recurrent Neural Network (RNN) \cite{58}, called the Reservoir. In case of spiking neural networks the reservoir is also called the Liquid Pool of neurons. In contrast to classic RNNs the connectivity inside the reservoir is set up at random and remains static, i.e. it is not adapted during learning.

2. Neurons that provide to the reservoir the external input $u(t)$ of the filter $F$ to be learned.

3. Neurons that constitute the output of the RC learning machine by combining inputs from the reservoir neurons. These output neurons are called Readouts. Their output time-series $\hat{y}(t)$ is an approximation $\hat{y}(t) \approx y(t)$ to the output $y(t)$ of $F$. The connections from the reservoir to the readouts are the only ones that are adapted during the (usually supervised) learning procedure.

In RNNs (like reservoirs) the neurons are linked together such that cyclic dependencies generally occur. That is, when displayed as a graph, the connectivity structure of the RNN exhibits cycles (or feedback loops), in the same way as graphical models may contain cycles in their representation of dependencies between variables (see figure \ref{fig:2.6}). Because connections between neurons are based on synapses (thick black arrows), which implies directed flows of information, a directed graph is needed for illustrating the network structure. The resultant interactions between nonlinear neurons lead to a further nonlinear transform of the input time series to some high-dimensional feature space, whose dimension is given by the number of neurons $N$ in the reservoir. Moreover, the presence of feedback loops allows the reservoir neurons to remain active for some period of time, even after input has ceased. This activity can hence be regarded as ‘echo’ of past inputs and thus implicitly endows the learning machine with memory. In total the reservoir serves two simultaneous purposes - nonlinear transformation along with memory of past inputs \cite{101,102}. The first purpose is typically referred to as the Kernel Property of the reservoir \cite{102}. Note however that in RC methods both, memory capacity and kernel property, are not explicitly built in, they rather emerge as an epiphenomenon of the reservoir’s randomly set up connectivity. Reservoirs that are useful in practice have finite memory capacity, i.e. the activity of the reservoir only reflects inputs that did not occur too far in the past. This property is typically referred as fading memory \cite{101,102}. By enabling feedback connections from the readout back to the reservoir (solid grey connections in fig \ref{fig:2.6}) it has been shown that memory capacity can be greatly enhanced \cite{99}.

The following section explains how these properties of the reservoir, together with the readout, can approximate any time-invariant filter with fading memory \cite{101}. Let $x_i(t)$ denote the activity (state) of the $i$-th reservoir neuron. A reservoir is said to have Echo States if, at any time $t$, its network activity vector $\mathbf{x}(t) = (x_1(t), ..., x_N(t))^T$ uniquely depends on the preceding input time series $u(t')$; $t' \leq t$ and not on sufficiently old states $\mathbf{x}(t')$; $t' \ll t$ \cite{66}. In this case $\mathbf{x}(t)$ can be described by a vector $\mathbf{E} = (E_1, ..., E_N)^T$. 

\begin{figure}[h] 
\centering 
\includegraphics[width=\textwidth]{example.png} 
\caption{Example figure for chapter 2.} 
\end{figure}
2.4. THE RESERVOIR COMPUTING FRAMEWORK

of filters that map input to output time series in a causal way:

\[ x(t) = [E(u^-)](t) \]  \hspace{1cm} (2.27)

where

\[ u^- : [\mathbb{R}, t] \rightarrow \mathbb{R} \]

\[ u^- (t') := u(t') \]

A reservoir having echo states can thus be interpreted as a filter bank, where each single neuron contributes to the bank a random and generally nonlinear filter \( E_i(u^-) \). The \( E_i(u^-) \) are referred to as Input Echo Functions [66] and depend on the used neuron model and the connectivity of the whole reservoir. As illustrated in figure 2.6, the readout neurons can access the filter bank by receiving input synapses from the neurons of the reservoir.

The readout constitutes a memoryless and generally (but not necessarily) linear device. That is, the filter outputs \( x_i(t) \) are linearly combined to form the instantaneous readout output \( \hat{y}(t) \):

\[ \hat{y}(t) = w^T \cdot x(t) \]  \hspace{1cm} (2.28)

where \( w \) is the vector of feedforward weights associated with the synaptic connections from the reservoir to the readout neurons. This vector can be adjusted using various different learning methods. In the simplest case, activity vectors \( x(t_k) \) are sampled at various time points \( t_k \) together with the corresponding values \( y(t_k) \) of the target filter to be learned (supervised learning). Subsequently the \( x(t_k) \) are arranged in a matrix \( X \) and, together with the \( y(t_k) \), \( w \) is determined by linear regression, e.g. by computing the pseudoinverse of \( X \). However as we will see in section 5.1 other regression methods based on \( X \) can be used as well. The conceptual elegance of RC methods is largely due to the ability to learn a nonlinear filter by training a set of feedforward weights, that is, by solving a convex, linear problem for which fast, effective solutions, both iterative and analytical, exist [20]. This stands in marked contrast to previous approaches for training RNNs [66], where the RNNs recurrent connectivity is learned [58, 126]. Training of the recurrent connections generally results in nonconvex optimization problems that are plagued by local minima of the associated cost function [66].

However the conceptual elegance of RC methods comes for a price: As a general purpose filter bank the reservoir must provide a rich set of input echo functions \( E_i(u^-) \), which endows the RC method with a large class of representable target filters [93]. It is apparent from equation 2.28 that this goal can be achieved most effectively by decorrelating the \( E_i(u^-) \) [100, 122]. Although for ESNs initial results in this
CHAPTER 2. BACKGROUND

Reservoir Readouts

Inputs

\[ u(t), x(t), y(t) \]

Readouts

\[ \hat{\ell}(t) \]

Figure 2.6: Architecture of the basic RC setup used in this thesis. Shown are the three conceptually distinct neural populations of a LSM/ESN and their corresponding connections. The black connections have to be present in all such setups, whereas the grey connections must exist only for implementations of specific filters. The input time series of such a filter is provided to the LSM/ESN through the thin, solid black connections. The letters above connections between populations indicate the output signal of the corresponding sending population (see text).

In case of ESNs however with tanh activation function, existence of echo states can be guaranteed by constraining the largest singular value \( \sigma_{\text{max}} \) of the connectivity matrix \( W \) by \( \sigma_{\text{max}} < 1 \) \[66\]. Also, it is empirically observed that it often suffices to scale the weight matrix such that \( |\lambda_{\text{max}}| < 1 \), where \( \lambda_{\text{max}} \) is the eigenvalue of \( M \) with the largest absolute value \[64\]. \( |\lambda_{\text{max}}| \) is also called the Spectral Radius.
Chapter 3

Results I: Abstract Functional Models of Spiking Processors
Implementing the Belief-Propagation Algorithm in an Event-Based Fashion

Using the frameworks explained in the previous chapter, we can now tackle the problems stated in the introductory part. More specifically, we can provide answers to the question of how the BP algorithm can be implemented using spike-based principles. Although spike-based, the principles presented in this chapter are abstract, in a sense that the processors that realize them are defined in purely mathematical terms disregarding entirely the use of neural elements (somas, dendrites, etc.). In other words, this chapter provides the abstract, functional descriptions that form the basis of the more concrete neural implementations of BP in chapter 5. We first describe the general event-based principle that underlies all processors in this chapter. We then lay out the main result of this thesis, namely the definition of a processor which is based on interspike-intervals (ISIs) and which allows for BP even in graphical models containing analog variables. This fact is demonstrated through results of direct software implementations of the processor. In this context it is also shown how in principle the processor’s spike output can be used for the unsupervised learning of the factor functions that make up the graph-parameterization. The chapter ends with the definition of a second event-based processor that allows for BP exclusively in
discrete graphical models.

3.1 A Meta-Principle for Event-Based Implementations of the Belief-Propagation Algorithm

In this section we lay out the fundamental principle behind all event-based BP-processors discussed in this thesis, i.e. the models presented in sections 3.2, 3.3 and 5. The core idea of these approaches is to exploit statistical meta-properties of distinct processors, that allow for the indirect computation of the Sum-Product-Rule (SPR). More concretely, the basic mathematical operations of the SPR (sums\integrals and products) are not implemented by specific modules of these processors, rather the SPR emerges as a statistical epiphenomenon of the latter's functionality. This is analogous to the spontaneous dynamics of systems in nature, which might also be considered as the result of complex computations on nature’s mathematically formulated laws.

Figure 3.1 illustrates the basic idea in an intuitive way. Although the presented nailboard model is computationally restricted (it only allows for gaussian dependencies between two variables), we can already see the two core ingredients that make the nailboard an efficient SPR processor:

1. The BP-messages are represented by a set of random samples, which in turn are defined by certain events (e.g. the event 'ball injected at position X').

2. The device’s functionality can be reduced to producing (random) output samples, whose distribution \( m(Y) \) is realized only by function evaluations (i.e. balls injected at sample position \( x \) trigger the ‘evaluation’ of function \( P(y \mid x) \) ).

These two ingredients allow for an exploitation of statistical rules that automatically realize the SPR without the processor’s need to implement sums, integrals or products. In the following, this will be laid out in more detail.

First, note that the mathematical structure of the SPR (equation 2.9) corresponds to the general rule for computing expected values of functions of random variables. If we define \( P(X_{\mathcal{N}(i) \setminus \{i\}}) := \prod_{k \in \mathcal{N}(i) \setminus \{j\}} \operatorname{m}_{k \rightarrow i}(X_k) \) to be a probability distribution of random vector \( X_{\mathcal{N}(i) \setminus \{i\}} \), then equation 2.9 computes the corresponding (conditional) expected value \( \mathbb{E}[f_i(X_{\mathcal{N}(i)}) \mid X_i = x_i] \). This is a very elegant property of the SPR, because it allows for the application of Monte-Carlo (MC) techniques to approximate the generally intractable integral in equation 2.9. MC methods can be used for such approximations if the integral to be solved can be expressed as the integral of an expected value computation of some random variable. In case of the SPR this condition is straightforwardly fulfilled if \( f_i(X_{\mathcal{N}(i)}) \) is taken to be that variable. The starting point of MC methods is a set of \( n \) independent,
identically distributed (i.i.d.) samples $S_X := \{x_1, ..., x_n\}$ of random variable $X$, drawn from distribution $P(X)$ (to avoid cluttered notation, we make use of the following definitions $X := X_{\mathcal{N}(i)} \setminus \{i_j\}$ and $Y := X_{i_j}$). In case of the SPR these samples form the basis of a sample set $S_{f_i}(y) := \{f_i(x_1, y), ..., f_i(x_n, y)\}$. The law of large numbers then assures that, in the limit of large $n$, the mean across $S_{f_i}(y)$ converges to $\mathbb{E}[f_i | Y = y]$ and therefore to the SPR.

The second elegant property of the SPR is the fact that $P(X)$ can be easily sampled from. Since $P(X) = \prod_{k \in \mathcal{N}(i) \setminus \{j\}} m_{k \rightarrow i}(x_{i_k})$ is a factorized distribution, a sample $x \in S_X$ is obtained by sampling from all $m_{k \rightarrow i}(x_{i_k})$ independently.

The resulting estimate of $\mathbb{E}[f_i | Y = y]$ is then used as a starting point for producing output samples of value $y$. This is precisely what all of the models in sections 3.2.1 and 3.3 have in common. They only differ in the way samples (or events) are encoded with spikes. The model presented in the next section is based on the simplest such encoding scheme: Every single spike corresponds to a single sample. However, as we will see in section 3.3 this is not the only useful strategy to define events with spikes.
Figure 3.1: Cartoon illustration of a nailboard model implementing a simple instance of the Sum-Product Rule.

Left column: Imagine a board with nails hammered in and on top of which a ball is placed and released at a single, distinct entry position $X = x$. As the ball falls through the forest of nails it bumps against them, leading to a random-walk behavior of the balls trajectory along the horizontal axis. Therefore, the horizontal position $Y$ where the ball exits the device very quickly becomes gaussian distributed, i.e. $P(y | x) = \mathcal{N}(x, \sigma)$ (bottom).

Middle column: If now, instead of placing the ball always at the same entry position $x$, a distribution $m(x)$ is used, how does the distribution of exit positions ($m(y)$) then look like? The answer is provided by Bayes’ rule: $m(y) = \sum_x P(y|x) \cdot m(x)$.

Right column: Because $m(y) = \sum_x P(y|x) \cdot m(x)$ is also an instance the SPR, the device’s functionality can be interpreted as computing an output message in a gaussian factor node (with $m(x)$, $m(x)$ denoting the input and output messages respectively).
3.2. ANALOG BELIEF-PROPAGATION USING INTERSPIKE INTERVALS

3.2 An Abstract Processor Based on Interspike-Intervals for Belief-Propagation in Analog Graphical Models

3.2.1 Functional Description of the Processor

Here we present the abstract, mathematical description of a spike based processor that can compute the SPR in the event-based fashion laid out in the previous section. The processor uses interspike-intervals (ISIs) as a means for defining events, i.e. the time elapsed between two consecutive spikes. This way of encoding BP-messages with spikes turns out to be the most general approach presented in this thesis, because it allows for neural implementations of BP in FFGs that even contain analog variables. For the sake of simplicity of exposition, the processor is described for output message computations of a factor node attached to three variable edges. However the proposed methods can be straightforwardly generalized to factors attached to an arbitrary number of variables.

In the case of a 3-way factor node $f$ attached to variables $X,Y,Z$ (see figure 3.2, left), equations 2.9-2.11 for an output message along $Z$ become:

\[
\begin{align*}
\text{uSPR}(z) & := \int_D \int_D f(x,y,z) \cdot m_X(x) \cdot m_Y(y) \, dx \, dy \quad (3.1) \\
N & := \int_D \text{uSPR}(z) \, dz \\
\text{SPR}(z) & := \frac{\text{uSPR}(z)}{N} \quad (3.3)
\end{align*}
\]

where we have replaced $um_{\text{out}}(z)$ and $m_{\text{out}}(z)$ by $\text{uSPR}(z)$ and $\text{SPR}(z)$ respectively for reasons which will become clear below.

At its core, the proposed processor is based on quantities that approximate scaled versions of $\text{uSPR}(z)$ and $N$ in equations 3.1 and 3.2 respectively. Hence we first show how these quantities can be obtained by a simple Monte Carlo method that, on a technical, non-algorithmic level, bears some similarities to Gibbs Sampling [52, 103]. The resulting estimates of $\text{uSPR}(z)$ and $N$ are then used to manipulate the firing statistics of an abstract spike-generation mechanism, which conveys $\text{SPR}(z)$ to a neighboring factor node.

In the proposed model, BP messages are represented by distributions of ISIs. Hence, the model employs a dedicated spike-generator whose firing statistics are controlled in a manner such that its ISI probability density function follows $\text{SPR}(z)$. We denote by letter $m$ the spike-based BP messages produced by the processor and the corresponding mathematically exact normalized and unnormalized messages by $\text{SPR}$ and $\text{uSPR}$ respectively (which fulfill equations 3.3 and 3.1 respectively). The value
m_X(x) message m_X assigns to value x of the analog variable X along which m_X is passed is therefore given by the generators probability density of firing two successive spikes with a temporal distance of x. This may be regarded as a firing of labelled spikes, where the analog label attached to each spike is given by the spikes preceding ISI (see figure 3.2a). According to the general strategy explained in section 3.1, this means that spikes are interpreted as random samples from their underlying ISI distribution, which permits approximating uSPR(z) and N by the following MC method (see figure 3.2b, right):

Starting from some position (x_0, y_0) on the (X, Y)-plane, each input spike from either m_X(x) or m_Y(y) updates that position according to the label it carries. For example, if the processor receives a spike from m_X(x) labelled by ISI value '3', then the position is updated from (x_0, y_0) to (3, y_0). Updates along the Y-axis are done in the same manner by spikes from m_Y(y). Assuming Z = z to be fixed for the moment, at each such sampled position (x, y) the factor function f_z(x, y) := f(x, y, z) is evaluated. Therefore one can associate a second kind of label with each spike, which is given by the result of such a function evaluation. In the following, this second label will be referred to as a 'function spike'. Evaluating all function spikes happening within a temporal sliding window [t - W, t] -at any time t during the message passing dynamics- and taking the sum across all of them yields a stochastic process S(z, t) approximately proportional to uSPR(z) in equation 3.1 (see section A.1 for proof). Replacement of f_z with the function F(x, y) := \int_D f(x, y, z) dz leads to a third spike label termed an 'integral function spike'. Summing all integral function spikes within W as before, one obtains a second stochastic process S_F(t) approximately proportional to N in equation 3.2 S(z, t) and S_F(t) are computed in parallel and with them all necessary quantities for computing SPR(z) in equation 3.3 are in place.

We now show how S(z, t) and S_F(t) can be used to construct a stochastic spike-generator whose ISI distribution m_out(z) is given by m_out(z) ≈ SPR(z). This step is important for a complete description of the abstract processor, because in general its output m_out(z) is used as spike input for a subsequent processor stage formed by a neighboring factor node. The spike-generator constitutes a nonstationary renewal system, i.e. at any time t subsequent spikes are fired randomly with a stochastic intensity, called hazard \[ h(t) = \frac{p(t - t_0, t)}{1 - \int_{t_0}^{t} p(t' - t_0, t') dt'} \]

(3.4)
If we associate value $z$ of variable $Z$ with $\Delta t$, i.e. set $z = \Delta t$, and want to have $p(\Delta t, t)$ to follow the SPR, we have to set

$$p(\Delta t, t) := \frac{S(\Delta t, t)}{S_F(t)} \approx \frac{r_{tot} \cdot W \cdot u_{SPR}(\Delta t)}{r_{tot} \cdot W \cdot N} = \text{SPR}(\Delta t)$$

(3.5)

where $r_{tot}$ is a constant explained in appendix 3.1 together with the approximation and why the latter steadily improves with increasing $W$. Plugging (3.5) into (3.4) yields a closed system for computing spiking BP-messages and completes the description of the abstract processor (see figure 3.3 for a block diagram of the complete, abstract processor). In the following section we show results of direct software implementations of the processor, i.e. implementations that disregard the use of any neural elements.

![Figure 3.2: Basic Principles of the Abstract ISI-based BP-Processor](image-url)

**a)** Spike encoding of BP messages: Each spike of a train representing a BP message is provided with an analog label, whose value corresponds to the length of the ISI preceding the spike, i.e. to the difference in spike times between the considered spike and its predecessor (see numbers above the spikes in arbitrary units). These analog values are therefore samples of the ISI distribution $p(x)$ underlying the spike train. A BP message $m_X(x)$ along variable $X$ is then defined by $m_X(x) := p(x)$.

**b)** 3-way factor node connected to analog variables $X, Y, Z$ and defined by some function $f(x, y, z)$. For all $z$, output message $m_{out}(z)$ is computed out of input messages $m_X(x)$ and $m_Y(y)$. The shown function $f_z(x, y)$ is given, for fixed $z$, by $f_z(x, y) := f(x, y, z)$. See text for details.
Figure 3.3: Block diagram of the abstract spiking BP processor

The following operations are executed at any time $t$ during execution of the processor:

1.) Positional Sampler: By means of a sliding summation window of length $W$, the positional sampler keeps track of a list of $n_W$ samples on the $(X, Y)$-plane. These samples are caused by input spikes from $m_X$ and $m_Y$ in $[t - W, t]$ and form the basis for computing both, function and integral function spikes.

2.) Integral Function Spike Labeler: The integral function spike labels only depend on $X$ and $Y$. Hence they are computed only once, as soon as their respective causing $(X, Y)$-sample comes in.

3.) Function Spike Labeler: In addition to the $(X, Y)$-samples, the function spike labels also depend on $Z$. If the spike-generator fires at time $t_0$ (see (6)), the function spike labeler is 'reset', i.e. the function spikes are labeled according to $f_z = 0$. Measuring $\Delta t = t - t_0$, the time passed since the last fired spike, these labels continuously change their values according to $f_z = \Delta t$.

4.) $\Sigma$: The quantities $S(\Delta t, t)$ and $S_F(t)$ are computed by summing all (integral) function spikes in $W$ respectively, leading to $5.)$ ISI probability density $p$ and cumulative distribution $\int_{t_0}^{t} p(t' - t_0, t') dt'$: Using $S(\Delta t, t)$ and $S_P(t)$, $p(\Delta t, t)$ is computed from equation 3.5 and accumulated by the integrator node ($f$). This integrator must be set to zero after each spike of the spike-generator.

6.) Hazard $h$: Both $p(\Delta t, t)$ and $\int_{t_0}^{t} p(t' - t_0, t') dt'$ are substituted into equation 3.4 to update $h(t)$.

7.) Spike Generator: Output spikes are drawn randomly at each time $t$ according to $h(t)$. Because of equation 3.4 it is assured that the resulting ISI distribution indeed satisfies the SPR (equation 3.5). To reset the function spike labeler and integrator, feedback of the fired spike is enabled.
3.2.2 Simulation Results of Non-Neural Implementations of the Abstract Processor

In this section we show results of direct software implementations of the abstract processor presented in the previous section. Although spike-based, these implementations are direct, in the sense that the functionality of figure 3.3 is realized by programs that entirely disregard the use of simulated neural elements (synapses, somas etc.) as an intermediate level. We will refer to this implementation as the black-box version of the processor - in later sections we will also consider more concrete neural versions that utilize such neuron-specific properties. We first evaluate the computational results of a single, isolated factor node attached to three variable edges, that is, three black-box processors that each compute their corresponding output message simultaneously. Consequently it is then verified that an interconnected network of such black-box processors can perform BP in full FFGs, i.e. that it can correctly compute the marginal probabilities of unobserved variables in the graph. Moreover, we show that our approach can also be used to perform inference over time by applying it to a Hidden Markov Model (HMM) [134, 15, 76]. Finally the processor’s functionality is slightly modified to qualitatively match ISI data from neurobiological experiments.

3.2.2.1 Computation of BP-Messages in a Single Factor Node

Figure 3.4 shows the single factor \( f(x, y, z) \approx f_{=} (x, y, z) = \delta(x - y) \cdot \delta(y - z) \) attached to variables \( X, Y, Z \) and chosen for basic evaluation of the abstract processor. Each of the corresponding three output messages is computed by a separate, independent processor, in response to three different, arbitrary ISI distributions representing the input messages. The approximation quality of the output messages with respect to results obtained by ordinary, discrete BP is determined. That is, equations 2.9, 2.10 are discretized and the resulting sums computed in a brute-force way. Although continuous in nature, due to their implementation in software the processors are necessarily advanced in discrete time steps, during each of which spikes are fired randomly according to the actual hazard (see section 3.2.1). All times are therefore measured as multiples of dimensionless time steps. The input ISI distributions contain probability mass only within the \( (0, 100] \) step interval. \( W \) is set to 750 steps. The factor function \( f(x, y, z) \) is an approximation to the equality constraint factor (see equation 2.3). Because the support of \( f_{=} \) is infinitesimally 'slim' around the line \( x = y = z \) - and hence cannot be hit by a sample with nonzero probability - for \( f \) its 'thickness' is increased using a Gaussian whose first principal component is set parallel to that line with infinite standard deviation. The standard deviations of the second and third principal component are set to 2 steps each.

The mathematical functions used for defining input messages \( m_{\text{in},X}, m_{\text{in},Y}, m_{\text{in},Z} \) are given in figure
3.4a. Application of the SPR to $f$ leads to $SPR(x) \approx m_{in,Y}(x) \cdot m_{in,Z}(x)$, $SPR(y) \approx m_{in,X}(y) \cdot m_{in,Z}(y)$, $SPR(z) \approx m_{in,X}(z) \cdot m_{in,Y}(z)$. For computing the controls, $f$ is discretized and stored in a 3-dimensional tensor of size $100 \times 100 \times 100$, representing all combinations of the first 100 values (time steps) of variables $X,Y,Z$. Note that this version of BP is very costly: For each of the 100 values of $SPR(z)$, $z \in (0, 100]$ a sum over $100 \times 100$ products has to be computed.

Figure 3.4 shows the ideal ISI distributions $SPR(\cdot)$ and their approximating $m_{out,(\cdot)}$. A close match between these two quantities can be observed.

Figure 3.4: Performance of computing output messages of a single factor node

a) The approximate equality constraint factor $f(x,y,z) \approx f_m(x,y,z) := \delta(x-y) \cdot \delta(y-z)$ depending on variables $X,Y,Z$. The ideal ISI histograms of the input messages $m_{in,X}(x) \propto |x - 50|$, $m_{in,Y}(y) \propto \sin \left(2\pi \frac{y}{100} \cdot y \right) + 1$, $m_{in,Z}(z) \propto e^{-\frac{z}{25}}$ are shown together with those of the corresponding ideal output messages $SPR(x) \approx m_{in,Y}(x) \cdot m_{in,Z}(x)$, $SPR(y) \approx m_{in,X}(y) \cdot m_{in,Z}(y)$, $SPR(z) \approx m_{in,X}(z) \cdot m_{in,Y}(z)$, as computed by brute-force, discrete BP. The actual output messages $m_{out,X}, m_{out,Y}, m_{out,Z}$ are computed by three processors following the dynamics described in section 3.2.1.

b) Top row: Magnified ideal output messages $SPR(\cdot)$ as in a.
Bottom row: Actual ISI histograms of the three processors computing the output messages $m_{out,(\cdot)}$. For each such histogram 3000 spike samples have been collected.
3.2. ANALOG BELIEF-PROPAGATION USING INTERSPIKE INTERVALS

3.2.2.2 Belief-Propagation in Full FFGs

We next consider how well a whole factor graph based on black-box processors can perform inference, i.e. can compute the beliefs given by equation 2.12. For that we investigate the two tree structures shown in figures 3.5a and 3.6a. In both cases, 30 trials with random instantiations of each involved factor function are conducted, i.e. during any such trial each factor function $f_a(x_a) = \sum_{i} w_i G_i(x_a | \mu_i, \Sigma_i)$ is given by an unnormalized mixture of Gaussians whose mixture components $G_i$ have random weights $w_i$, mean vectors $\mu_i$ and covariance matrices $\Sigma_i$ (see appendix section A.2 for parameters). Therefore, each trial represents an arbitrary inference problem modeled as FFG. Variables of the FFG could take on integer step values in $(0, 130]$. After having collected 3000 spike-samples for each message in each of the 30 inference runs, we compute the beliefs $B(x_{ij})$ for some variables $x_{ij}$ using the message ISI histograms and equation 2.12 (with the integral replaced by a sum across discrete time steps). Again, as a control, the true marginals $P(x_{ij})$ are computed by ordinary, discrete BP. Note that in figure 3.6 we have to coarsen the time resolution from 1 to 4 steps to keep the task tractable for the ordinary BP method. Otherwise tensors of size $130^4 \approx 2.86 \cdot 10^8$ would have to be stored and 130 sums each over $130^3$ products processed. In contrast, the black-box processors are still run at a resolution of 1 step, but to keep comparability their ISIs are also binned using the coarser time grid.

To determine the approximation quality, the ordinary $D(B(P))$ and a normalized $D_{norm}(B(P) := \frac{D(B(P)}{H(B,P)}$ Kullback-Leibler (KL) divergence [27] between $B(x_{ij})$ and $P(x_{ij})$ are calculated, where $H(B,P)$ denotes the cross-entropy $H(B,P) := \sum_{x_{ij}} B(x_{ij}) \cdot \log (P(x_{ij}))$ between $B$ and $P$. Hence $D_{norm}(B(P)$ is the fraction of additional (wasted) bits used by a code of length $H(B,P)$ constructed from $P(x)$, compared to the minimum code length $H(B)$. Figures 3.5b,c and 3.6b,c show the corresponding results. It can be seen that the fraction of wasted bits is at most around 5% and below 1% in most cases, that is, typically very small.

To evaluate chance level, in figures 3.5c and 3.6c we compare the values of $D(B(P))$ to their 3σ-range, when $B(x_{ij})$ is replaced by a distribution $B^*(x_{ij})$ that is drawn from a uniform distribution of distributions of $x_{ij}$. The 3σ-range is given by those values of $D(B^*(P))$ that are within three standard deviations $\sigma [D(B^*(P))$ around the expected value $\mathbb{E} [D(B^*(P))$. It is apparent from these figures that the performance of $B(x_{ij})$ as found by the processors is far better than the average performance of randomly drawn distributions $B^*(x_{ij})$.

Due to the coarser time resolution and the corresponding smaller number of states $n$ per variables, the 3σ-ranges in figure 3.6c are somewhat larger than in figure 3.5c (see section B). In some cases these ranges almost include $D(B^*(P) = 0$, which leads to long error bars in log-plots.
Figure 3.5: Computing marginals in a randomly parameterized FFG I

a) FFG tree structure used as starting point for 30 inference trials with factor functions A1-E2 randomly defined by gaussian mixtures (see text for details). The maximum branching degree of a factor node is 3.

b) Top: Raw data of the belief of variable \( X \) connecting factor nodes C1 and C2 for trial number 2 as found by the black-box implementation of the abstract processor. Bottom: Exact marginal distribution \( P(x) \) of variable \( X \) (blue bars). This is the result of ordinary, discrete BP. The red curve shows the moving averaged version \( B(x) \) of the belief in the top row, averaging window size was 5 points.

c) x-axis: Trial number, y-axis: Values of \( D(B \| P) \) (black dots, in bits) and \( D_{norm}(B \| P) \) (red dots) for representative variables. Blue bars indicate the 3σ-range of \( D(B^* \| P) \) if instead of \( B(x_i) \) a random distribution \( B^*(x_i) \) is taken (see main text and appendix for details).
3.2. ANALOG BELIEF-PROPAGATION USING INTERSPIKE INTERVALS

Figure 3.6: Computing marginals in a randomly parameterized FFG II
a) FFG tree structure used as starting point for 30 inference trials with factor functions A1-D2 randomly defined by gaussian mixtures (see text for details). The maximum branching degree of a factor node is 4.
b) Top: Raw data of the belief of $X$ as found by the black-box processing method. Bottom: Exact marginal distribution $P(x)$ of variable $X$ connecting factor nodes $B_1$ and $C$ for trial number 9 (blue bars). This is the result of ordinary, discrete BP. The red curve shows the moving averaged version $B(x)$ of the belief in the top row, averaging window size was 3 points.
c) x-axis: trial number, y-axis: Values of $D(B \parallel P)$ (black dots, in bits) and $D_{\text{norm}}(B \parallel P)$ (red dots) for representative variables in each trial. Blue bars indicate the $3\sigma$-range of $D(B^\ast \parallel P)$ if instead of $B(x_{ij})$ a random distribution $B^\ast(x_{ij})$ is taken (see main text and appendix for details).

3.2.2.3 Computation of BP Messages With Biologically Realistic ISI Distributions

So far, the ISI distributions representing BP messages were of arbitrary type, i.e. they could be described by any nonnegative function that is normalized to 1. Experimental results from measurements in cortex however suggest that activity of real, biological neurons rather leads to a restricted set of ISI distributions, that are quite dissimilar to many of those in figures 3.4, 3.5 and 3.6. Here we show how the abstract processor description can be slightly modified to respect these biological findings while, at the same time, the full computational power of the processor is preserved.

As discussed in more detail in the chapter summary, many studies have reported consistency of ISI
distributions with a class of unimodal distributions (quasi-exponential, gamma and log-normal distributions) in a variety of cortical areas. These results are usually taken as indications for quasi-Poissonian firing and so point to a rather restricted class of ISI distributions in cortex. Figure 3.7a shows a typical experimental ISI histogram on which this claim is based. However, if histograms of the log(ISI) rather than the ISI values are formed, the results become quite inconsistent with the Poisson assumption \[138, 153\] (fig.3.7c). This finding holds true even when the corresponding ISI distributions appear quasi-exponential (see \[138\], figure 1). Log-ISI histograms in contrast make features apparent that are barely visible in linearly scaled histograms \[153\].

We now lay out in detail how the abstract processor can be modified to become qualitatively consistent with such experimental data. Imagine the ISI spike labeling of fig. 3.2a being replaced by corresponding labels of the function \(g(ISI) := \log(ISI)\). The sampling process of fig. 3.2b is then performed for some function \(f_{z^*}(x^*, y^*)\) of the transformed variables \(x^* := g(x), y^* := g(y), z^* := g(z)\) (in the following an asterisk '*' indicates either quantities in the transformed space, or functions thereof). As before, the stochastic processes \(S(z^*, t)\) and \(S_F(t)\) are subsequently computed from these samples to form \(p_{z^*}(z^*, t) := \frac{S(z^*, t)}{S_F(t)}\) (equation 3.5). However, to achieve consistency between the log encodings of the input and output we have to ensure that \(p_{z^*}(z^*, t)\) represents a distribution of log(ISI) rather than ISI values. Therefore, the ISI distribution \(p(\Delta t, t)\) that is used to update the hazard (equation 3.4) has to be chosen such that the corresponding log(ISI) distribution is given by \(p_{z^*}(g(\Delta t), t)\). This can be done by referring to the distribution of (monotonic) functions of a random variable \[124\]:

\[
p(\Delta t, t) = \frac{p_{z^*}(g(\Delta t), t)}{\left| \frac{d g^{-1}(g(\Delta t))}{d z^*} \right|} = \frac{S(\log(\Delta t), t)}{S_F(t) \cdot |\Delta t|} \tag{3.6}
\]

Incorporating these two aspects in the abstract processor description -sampling in the log(ISI) domain together with replacement of equation 3.5 by equation 3.6- leads to a modified processor, whose log(ISI) distribution represents a BP message. The question is then whether the resulting ISI distributions are indeed closer to the experimental data. In this context, similarly to figure 3.4, fig.3.8 shows simulation results of a single factor node that receives arbitrary, log(ISI) encoded input messages. By consistently displaying a prominent peak at small ISI values and a decaying envelope, the resulting ISI distributions are indeed more similar to exponential distributions. In (b) and (c) some of the highly structured log(ISI) distributions correspond to ISI distributions that are rather simple in nature (see for example the red and blue rows in (b) and (c) respectively). Interestingly, this feature is exactly what is observed experimentally in area V1 in macaque monkeys \[138\], figure 1), see also \[138, 153\] for a variety of structured log(ISI)
In summary, by switching from ISI to log(ISI) encoding of BP messages, the abstract processor can qualitatively reproduce features of experimental ISI distributions. Moreover, the processor provides an algorithmic meaning to the nontrivial structure observed in experimental log(ISI) distributions. We have shown that such structure can be reproduced using our spike-based BP processor, even if the corresponding ISI distribution appears quasi-exponential and hence simple.

![Graphs](image-url)

**Figure 3.7: Example Experimental ISI and log(ISI) histograms**

a) Rate parsed ISI histogram of a neuron from area MST in macaque cortex during observation of a moving dot stimulus. The cell exhibits irregular, quasi-Poissonian spiking behavior, indicated by the quasi-exponential shape of the histogram, whose coefficient of variation is $CV = 1.07$. Parsing is done for rates between 15 and 20 Hz, the red line represents the best fitting gamma distribution for the histogram (figure taken from [104]).

b) Rate parsed ISI histogram of an area 5 neuron in macaque cortex during observation of a moving dot stimulus. The distribution is inconsistent with a Poisson process due to $CV = 0.73$. Parsing is done for rates between 30 and 35 Hz. The red line is represents the best fitting gamma distribution for the histogram (figure taken from [104]).

c) Upper plot: Histogram of the log(ISI) values of a neuron in cat primary auditory cortex during presentation of a ripple stimulus. The diagram exhibits two prominent local maxima, located in two distinct regions of the histogram, the manually selected small and large ISI regions (grey shaded regions). Lower plot: log(ISI) histogram of a simulated linear-nonlinear Poisson model neuron [150], whose linear filter is given by the estimated spectrotemporal receptive field of the neuron in the upper plot. Together with a spiking nonlinearity this filter determines the instantaneous firing rate of the model neuron, yielding an inhomogeneous Poisson spike process. Clearly, with the Poisson assumption one cannot reproduce the essential features, especially in the small ISI regime (figure taken from [153]).
Figure 3.8: Performance of a single factor node computing log(ISI) encoded messages.

a) The approximate equality constraint factor
\[ f(x^*, y^*, z^*) \approx \tilde{f}(x^*, y^*, z^*) := \delta(x^* - y^*) \cdot \delta(y^* - z^*) \]
depending on variables \( X^*, Y^*, Z^* \). With \( a := \log(10), b := \log(151) \) spikes are labeled according to \( \log(ISI) \). The corresponding distribution is termed the log(ISI) distribution. The ideal log(ISI) distributions of the input messages
\[ IN_{X^*}(x^*) = |x^* - \frac{1}{2}(b-a)|, IN_{Y^*}(y^*) = \sin\left(2\pi \frac{x^*}{b-a} \cdot y^*\right) \]
and \( IN_{Z^*}(z^*) = \exp\left[-\left(z^* - \frac{1}{4}(b-a)\right)^2\right] \)
are shown together with those of the corresponding ideal output messages \( SPR(x^*) \approx IN_{Y^*}(x^*) \cdot IN_{Z^*}(x^*), SPR(y^*) \approx IN_{X^*}(y^*) \cdot IN_{Z^*}(y^*), SPR(z^*) \approx IN_{X^*}(z^*) \cdot IN_{Y^*}(z^*) \), as computed by brute-force, discrete BP.

b) Left column: Magnified histograms of the ideal input messages \( IN_{\{i\}}(\cdot) \) as in (a). Middle column: Real log(ISI) histograms of modified, abstract processors. Right column: ISI histograms of the same spike data as in the middle column. These ISI distributions constitute the actual, physical input to the processors.

c) The same as in (b) for the corresponding output messages.
3.2. ANALOG BELIEF-PROPAGATION USING INTERSPIKE INTERVALS

3.2.2.4 Inference Over Time

This section is intended to show that the abstract ISI processor cannot only be applied to static inference problems, but also to cases where changing environmental observations are absorbed over time. Hence we show some initial results for inference in a Hidden Markov Model (HMM), which in contrast to the previous cases amounts to BP message-passing in a loopy graph. We consider a setting similar to [17], where model predictions about a moving stimulus are compared to corresponding audio-visual sensory cues. For a simple scenario the processors ability to perform dynamic inference over time is shown. Thereby our simulations also confirm that the Monte-Carlo approximation 3.5 does not lead to error propagation, at least in the considered small, loopy graph.

Börlin and Denève have considered the task of dynamically estimating the position \( x_t \) of a moving one-dimensional stimulus, that evolves according to the following drift-diffusion dynamics [17]

\[
\frac{dx_t}{dt} = \delta dt + \sigma dW_t \tag{3.7}
\]

where \( x_t \) is the position of the stimulus at time \( t \), \( \delta \) is the drift and \( \sigma \) the diffusion parameter and \( W_t \) is a Wiener process. We consider the time-discrete version of 3.7

\[
x_{t+1} = x_t + \delta + \xi_t \tag{3.8}
\]

where the \( \{\xi_t \mid \xi_t \sim \mathcal{N}(0, \sigma^2), t \in \mathbb{N}\} \) are mutually independent gaussian noise variables. The dynamics defined by 3.8 can be captured by a simple gaussian factor node \( P(x_{t+1} \mid x_t) \) relating \( x_{t+1} \) and \( x_t \) through

\[
P(x_{t+1} \mid x_t) = \mathcal{N}(x_t + \delta, \sigma^2) \tag{3.9}
\]

This factor node can be regarded as an agents internal model about the stimulus movement and, upon combination with auditory and visual sensory cues, it allows the agent to predict the position of the stimulus. Denote with \( A_t / V_t \) respectively the observed auditory and visual cue at time \( t \). We consider \( A_t \) and \( V_t \) to be noisy versions of \( x_t \), i.e.

\[
P(A_t \mid x_t) = \mathcal{N}(x_t, \sigma_A^2)
\]

\[
P(V_t \mid x_t) = \mathcal{N}(x_t, \sigma_V^2) \tag{3.10}
\]

3.9 and 3.10 define a HMM whose Bayesian network form is shown in figure 3.9. The model can be used as starting point for computing the posterior distribution \( P(x_{t+1} \mid A_0, \ldots, A_t, V_0, \ldots, V_t) \), i.e. the distribution of \( x_t \) given all sensory evidence absorbed so far. For that, we first translate the BN into a
corresponding FFG and then emulate the latter using our setup of interconnected ISI processors (figure 3.9b).

The processors use the log(ISI) spike labeling scheme of figure 3.8 hence all variables of the HMM are defined in log(ISI) space. We consider the simplified setup of temporally constant observations $\forall t : A_t = A_0, V_t = V_0$. Figure 3.10 shows, for two arbitrary settings of $A_0$ and $V_0$, the development of the posterior mean $\mu_{[X_{t+1} | A_0]}$ and standard deviation $\sigma_{[X_{t+1} | A_0]}$ (see appendix (section A.2 for the model parameters used in this simulation). As for the single-node simulation (fig. 3.4), we had to increase the 'thickness' of the support of the equality constraint factor nodes. Therefore one cannot not be sure if the posterior decoded from the spikes is indeed close to the true posterior, since changing the factor functions might change the whole factorized representation. This problem persists even if the decoded results match those of the corresponding brute-force simulation. Fortunately however, in case of constant observations and gaussian messages it is possible to apply the methods of [91] to theoretically compute the steady-state values $\mu_{ss}, \sigma_{ss}$ of $\mu_{[X_{t+1} | A_0]}$ and $\sigma_{[X_{t+1} | A_0]}$ respectively:

$$
\mu_{ss} = \frac{A_0 \sigma_V^2}{\sigma_A^2 + \sigma_V^2} + \delta \left( \frac{\sigma_A^2 \sigma_V^2 + \frac{1}{4} \sigma^4}{\sigma^2} + \frac{1}{2} \right) 
$$

$$
\sigma_{ss} = \sqrt{\frac{\sigma_A^2 \sigma_V^2 \sigma^2}{\sigma_A^2 + \sigma_V^2} + \frac{1}{4} \sigma^4 + \frac{1}{2} \sigma^2}
$$

Computation of these quantities reveals that a network of ISI processors can perform inference over time, since the spike-decoded posterior indeed converges towards nearby values (figure 3.10). It is also evident that the necessary modifications of the equality constraint factors have no substantial impact on the results (compare the dashed and the dashed-dotted line). Note however that, by moving to discrete time in equation 3.8 we have implicitly decoupled the time-base of the HMM from physical time, on which message-passing plays out due to the dynamics of the ISI processor. This aspect does not matter for the considered case of temporally constant observations, where steady-state analysis is sufficient. In a more intricate setup however that entails the dynamic accumulation of changing sensory evidence, it is important to mathematically express the meaning of a message for its underlying HMM variable (e.g. $X_{t+1}$) at any given moment in physical time. We consider the presented results as an initial step towards this more general case and leave the issue regarding the two time-scales for future work.

In summary, this section has shown that in principle a setup of interconnected ISI processors cannot only be used for static inference tasks, but also for tasks involving inference over time in a corresponding loopy graph. This holds true despite the danger of error-propagation in such graphs. Although we have examined only a simple FFG with constant input messages (observations), our initial results make us
confident that the ISI processor can perform inference over time also in more complicated setups.

Figure 3.9: Inference over time
a) A Bayesian Network HMM describing how auditory ($A_t$) and visual ($V_t$) sensory cues depend on a evolving hidden state ($X_t$).
b) Functional FFG-style representation of the HMM in (a). The arrows correspond to spike-based messages computed by abstract ISI processors. Only the messages represented by arrows are computed. The graph is not a true FFG of the BN in (a), since it does not explicitly represent the BNs factorization over time. The red message is the desired posterior probability $P(X_{t+1} | A_0, ..., A_t, V_0, ..., V_t)$
Figure 3.10: Predicted stimulus $x_{t+1}$ for two trials with constant observations $A_0 = A_1 = ... = A_t$ and $V_0 = V_1 = ... = V_t$

a) $A_0 = 0.64, V_0 = 0.74$; Top: Development of the posterior mean as decoded from the processors’ spike trains. The time axis was sampled at 4000 regular intervals, and for each such sampled time point $t_0$ the mean ISI label of all spikes in $[t_0 - 500, t_0]$ is plotted (blue trace). The dashed line shows the steady-state value of the posterior mean as found by brute-force, discrete BP. The corresponding theoretical steady-state value $\mu_{ss}$ (equation 3.11) is virtually identical.

Bottom: Development of the posterior standard deviation. The dashed line shows the brute-force steady-state value and the dashed-dotted line the corresponding theoretical value $\sigma_{ss}$ (equation 3.12).

b) The same as in (a) for $A_0 = 0.56, V_0 = 0.60$

3.2.3 Implications of the Abstract Processor For Learning in Analog Graphical Models Using Expectation-Maximization

So far we have only considered the case when the function defining a factor node, i.e. $f_i(x_{NV(i)})$ in eq.2.9 was predetermined. In a biologically realistic setting however these functions have to be learned through experience, that is from a training set of data points. Here we show in principle how ISI sample output of the spike-based BP processor can be used to perform such learning in an unsupervised fashion.

The learning framework we employ is Expectation-Maximization (EM) [31] applied to analog graphical models. The presented results are exclusively on a conceptual\mathematical level, no simulations that do apply these concepts in practice have been conducted so far. Also, we do not give any account on how these methods may be implemented in concrete biological systems, rather, they possibly provide an abstract, functional and algorithmic framework for such systems. Reading this section requires some basic, previous knowledge about the EM-algorithm in general [31, 15] and about its application to graphical models in particular [31, 45, 143]. Our results are presented along the lines of Rolfe [143], and we will
specifically show how the (discrete) EM update rule described there can be mimicked in case of analog FFGs with a sample-based representation of messages.

When graphical models are considered in a biological setting, one question inevitably arises: Even if inference in some graph enables a biological agent to sensibly interact with its environment, how does the agent obtain this graphical representation of dependencies between internal and external, environmental variables (see fig. 2.3 for example)? Given the abundance of environmental scenarios the agent might be put it, the sequence of base-pairs in a DNA-string is not long enough for genetic encoding of the neuronal connectivity that is necessary to physically support each such graph. Also, purely genetic predetermination would render an agent incapable of adapting to environmental changes, which necessarily require updates of its model of the world. Alternatively, assuming some fixed connectivity structure of the graph, the factor functions can be learned from the data the agent receives from its various sensory modalities (visual, auditory, etc.). In the following we will use the term parameterization for any complete instantiation of parameters of each factor function in a graph. We will look at two different ways for defining such parameterizations, each of which will give rise to slightly different methods for EM-learning.

Let \( f_a(X_a) \) be the factor function of factor node \( a \) that is supposed to be updated during the M-step of the EM learning procedure. \( X_a := X_{N_v(a)} \) is the shorthand notation for the vector of variables \( f_a \) depends on. Let \( \langle P_{\text{obs}}(X_a | Y) \rangle := \frac{1}{m} \sum_{j=1}^{m} P_{\text{obs}}(X_a | Y^j) \) be the mean marginal probability of \( X_a \) conditioned on the vector of observed variables \( Y^j \) across all training data \( j \in \{1, \ldots, m\} \). Define \( P_{\text{free}}(X_a) \) to be the unconditioned marginal, i.e. when inference in the graphical model is free-running without any variables observed. Hence the quantities \( P_{\text{obs}}(X_a | Y^j) \) and \( P_{\text{free}}(X_a) \) can all be determined by means of BP. As laid out by Ghahramani [56], the goal of the M-step is to achieve equality between the mean conditional and the unconditional marginals:

\[
\forall a : P_{\text{free}}(X_a) = \langle P_{\text{obs}}(X_a | Y) \rangle 
\tag{3.13}
\]

Whereas the \( \langle P_{\text{obs}}(X_a | Y) \rangle \) have been determined during the E-step -and are supposed to stay constant during the whole subsequent M-step- the \( P_{\text{free}}(X_a) \) continue to change as the parameterization is updated in the M-step. The approximate (discrete) EM-update rule in [143] softens this strict division of responsibilities among E- and M-steps, by updating the parameters of a single factor only before the next \( \langle P_{\text{obs}}(X_a | Y) \rangle \) are estimated. From this rule it follows that, after an update of \( f_a \), the \( P_{\text{free}}(X_a) \) are equal to the \( \langle P_{\text{obs}}(X_a | Y) \rangle \) before the update. The rule then tries to fulfill condition \( \text{3.13} \) by repetitive matchings of these two quantities for the different factors \( f_a \) in the graph. Once the parameterization has converged, condition \( \text{3.13} \) is fulfilled for all the factors and hence defines a termination criterion for the whole learning algorithm. We will show later two methods for a single factor update, i.e. for achieving
equality between the old \( P_{\text{obs}}(X_a|Y) \) and the new \( P_{\text{free}}(X_a) \) in case of analog FFGs using sample-based BP-messages. It is thus possible to emulate the approximate EM-rule of [43] also in the analog general case. Since we use BP as a means of inference, the two quantities of interest are computed from the input messages \( m_{k\to a}(x_{ak}) \) of factor \( f_a \) in the following way:

\[
P_{\text{free}}(x_a) = \frac{1}{Z_{\text{free}}} \cdot f_a(x_a) \cdot M_{\text{free}}(x_a)
\]

\[
P_{\text{obs},j}^*(x_a|y^j) = \frac{1}{Z_{\text{obs},j}^*} \cdot f_a^*(x_a) \cdot M_{\text{obs},j}(x_a|y^j)
\]

\[
M_{\text{free}}(x_a) := \prod_{k \in \mathcal{N}(a)} m_{k\to a}(x_{ak})
\]

\[
M_{\text{obs},j}(x_a|y^j) := \prod_{k \in \mathcal{N}(a)} m_{k\to a}(x_{ak}|y^j)
\]

\[
Z_{\text{free}} := \mathbb{E}[M_{\text{free}}[f_a]]
\]

\[
Z_{\text{obs},j}^* := \mathbb{E}[M_{\text{obs},j}[f_a^*]]
\]

where

\[
\mathbb{E}_P[g] := \int_{\mathcal{D}^{n_a}} g(x_a) \cdot P(x_a) \, dx_{a_1} \cdots dx_{a_{n_a}}
\]

is the expected-value of function \( g(x_a) \) with respect to distribution \( P(x_a) \). Quantities marked with an asterisk \('*'\) are considered constant during a single update and have been determined prior to the update. For example, \( f_a^*(X_a) \) is not affected by an update in contrast to \( f_a(X_a) \). Once a single \( f_a \) has been modified and \( P_{\text{free}}(X_a) = \langle P_{\text{obs}}^*(X_a|Y) \rangle \) holds true, the input messages of another factor \( f_{a'} \), \( a' \neq a \), that is updated in the next iteration of the algorithm, are recomputed. Recomputation of these messages is needed, because they mediate the dependence of \( P_{\text{free}}(X_a') \) and \( \langle P_{\text{obs}}(X_{a'}|Y) \rangle \) on \( f_a \). This sequence of alternating message computations and factor updates is then repeated until convergence.

As we will see below, the product-messages \( M_{\text{free}}(X_a) \) and \( M_{\text{obs},j}(X_a) \) are needed for updating factor \( f_a \). In this context, a problem arises with the input messages \( m_{k\to a}(x_{ak}) \): Since our spike-based BP processor represents messages only indirectly as lists of ISI-labeled spike-samples \( s_{k\to a} = \{x_{ak}^1, x_{ak}^2, x_{ak}^3, \ldots \} \), how then can \( M_{\text{free}}(X_a) \) and \( M_{\text{obs},j}(X_a) \) be properly defined? A straightforward solution to this problem is to take the cartesian product of all \( n_a \) message sample lists to obtain sets of samples from \( M_{\text{free}}(X_a) \) and \( M_{\text{obs},j}(X_a) \) respectively:

\[
S_{\text{free}} := \times_{k \in \mathcal{N}(a)} s_{k\to a}, \quad S_{\text{obs},j} := \times_{k \in \mathcal{N}(a)} s_{k\to a, j}
\]
These sets then respectively allow for a substitution of $M_{free}(X_a)$ and $M_{obs,j}(X_a)$ in the factor update equations.

We now derive two such updating methods, a parametric and a nonparametric one, that both aim at fulfilling $P_{free}(X_a) = \langle P_{obs}^*(X_a | Y) \rangle$ for a given factor $f_a$. Both methods utilize a sample-based representation of messages, however, they differ in the way factor functions are represented and their corresponding updates are performed.

3.2.3.1 Nonparametric EM

Here we consider updates of a factor function $f_a$ that is represented nonparametrically. That is, based on some set of samples, $f_a$ is supposed to be given by a corresponding kernel density estimate \[15\]. In brief, these methods place a stereotypical kernel function $K(x_a)$, e.g. a Gaussian, at the position of each sample. An estimate $\hat{P}(x_a)$ of the true underlying probability density $P(x_a)$ of the samples can then be obtained by the sum of all such kernels. For example, assume the set $S_{free}$ with $N$ samples be given. The corresponding kernel density estimate $\hat{M}_{free}(X_a) \approx M_{free}(X_a)$ is then defined by:

$$\hat{M}_{free}(x_a) = \frac{1}{N} \sum_{x_a^f \in S_{free}} K(x_a - x_a^f) \quad (3.22)$$

where we have assumed $K$ to have a volume normalized to 1. Correspondingly, we could also estimate the $M_{obs,j}(X_a), j \in \{1, \ldots, m\}$ based on the sample sets $S_{obs,j}$. In the present context however, these sets form the basis for an estimate $\hat{P}_{obs}^*(X_a)$ of $\langle P_{obs}^*(X_a | Y) \rangle$. Following eq.3.15 this is achieved by weighting the kernels appropriately:

$$\hat{P}_{obs}^*(x_a) = \frac{1}{Z_{obs,j}} \sum_{j=1}^{m} \sum_{x_a^o \in S_{obs,j}} f_{a}^*(x_a^o) \cdot K(x_a - x_a^o) \quad (3.23)$$

with $Z_{obs,j} := \sum_{x_a^o \in S_{obs,j}} f_{a}^*(x_a^o)$. The two functions 3.22 and 3.23 can then be used for redefining (updating) $f_a(x_a)$:

$$f_a(x_a) = \frac{\hat{P}_{obs}^*(x_a)}{\hat{M}_{free}(x_a)} \quad (3.24)$$

It can be straightforwardly verified that, if the true product-messages $M_{free}(X_a)$ were known, inserting 3.24 into 3.14 yields an approximation to $\langle P_{obs}^*(X_a | Y) \rangle$ as desired.

We now turn to a different approach for achieving $P_{free}(X_a) = \langle P_{obs}^*(X_a | Y) \rangle$, where a parametric model rather than a set of samples is used for defining the factor function.
3.2.3.2 Parametric EM

Here we assume at first \( f_a (X_a) \) to be given in parameterized form, e.g. as a mixture of Gaussians (see section 3.2.2). The basic idea is then to fulfill \( P_{\text{free}} (X_a) = \langle P_{\text{obs}}^* (X_a | Y) \rangle \) by fitting \( P_{\text{free}} (X_a) \) -which depends on \( f_a (X_a) \) via eq. 3.14- to a set of samples from \( \langle P_{\text{obs}}^* (X_a | Y) \rangle \) by means of log-likelihood maximization. After inference has been performed, we do not have such samples ready to use, however, we can equivalently take the sample sets \( S_{\text{obs}, j} \), which we have by then, and maximize the following weighted log-likelihood:

\[
\mathcal{L} (\theta) := \sum_{j=1}^{m} \frac{1}{Z_{\text{obs}, j}} \sum_{x_a^j \in S_{\text{obs}, j}} f_a^* (x_a^j) \cdot \log (P_{\text{free}} (x_a^j)) = \sum_{j=1}^{m} \frac{1}{Z_{\text{obs}, j} (\theta^*)} \sum_{x_a^j \in S_{\text{obs}}} f_a (x_a^j | \theta^*) \cdot \left( \log f_a (x_a^j | \theta) + \log M_{\text{free}} (x_a^j) - \log Z_{\text{free}} (\theta) \right)
\]

where \( Z_{\text{obs}, j} := \sum_{x_a^j \in S_{\text{obs}, j}} f_a^* (x_a^j) \approx Z_{\text{obs}, j} \) is the Monte-Carlo estimate of \( Z_{\text{obs}, j} \). In the second line we have made explicit the dependence of \( f_a \) and \( Z_{\text{free}} \) on the local parameter vector \( \theta \) that controls the parameterization of \( f_a \), e.g. the weights, means and covariance matrices of a gaussian mixture. \( \theta^* \) denotes the parameter vector as it was during computation of the messages, e.g. \( f_a (x_a^j | \theta^*) = f_a^* (x_a^j) \).

Correspondingly, \( \theta \) is the parameter vector that is modified in the subsequent update. To perform the update, i.e. to fit \( P_{\text{free}} (X_a) \) to a set of samples from \( \langle P_{\text{obs}}^* (X_a | Y) \rangle \), gradient-based optimization techniques can be used to maximize eq. 3.25. A variety of methods are available for that purpose, e.g. the Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton method [12]. As these techniques rely on gradient information, we need to provide the gradient for our problem at hand:

\[
\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{j=1}^{m} \sum_{x_a^j \in S_{\text{obs}}} \frac{1}{Z_{\text{obs}, j}} \cdot \frac{f_a^* (x_a^j)}{f_a (x_a^j | \theta)} \cdot \frac{\partial f_a (x_a^j | \theta)}{\partial \theta} - \frac{1}{Z_{\text{free}} (\theta)} \cdot \mathbb{E}_{M_{\text{free}}} \left[ \frac{\partial f_a (x_a | \theta)}{\partial \theta} \right]
\]

Note that, for all functions \( g(x_a) \), the quantity \( \mathbb{E}_{M_{\text{free}}} [g] \) can be obtained by the Monte-Carlo approximation \( \approx \sum_{x_a^j \in S_{\text{free}}} g (x_a^j) \). Likewise \( Z_{\text{free}} (\theta) \) is approximated through

\[
Z_{\text{free}} (\theta) \approx \hat{Z}_{\text{free}} (\theta) := \sum_{x_a^j \in S_{\text{free}}} f_a (x_a^j | \theta)
\]

These estimates then allow for computation of the gradient, that is subsequently used by the chosen optimization technique for solving the fitting problem.

A second parametric method achieving \( P_{\text{free}} (X_a) = \langle P_{\text{obs}}^* (X_a | Y) \rangle \) is very similar to the previously described nonparametric approach. However, rather than relying on nonparametric density estimates, the former is based on parametric models of the densities \( \langle P_{\text{obs}}^* (X_a | Y) \rangle \) and \( M_{\text{free}} (X_a) \). Using the
3.2. ANALOG BELIEF-PROPAGATION USING INTERSPIKE INTERVALS

sample set \( S_{\text{free}} \) one can find an estimate \( \hat{M}_{\text{free}}(X_a) \) of \( M_{\text{free}}(X_a) \) by maximizing the log-likelihood of the parametric model, with methods more efficient than gradient-based techniques. For example, one can rely on an EM-procedure as described in Bishop ([15], chapter 9). The estimate \( \hat{P}_{\text{obs}}(X_a|Y) \) of \( \langle P_{\text{obs}}^*(X_a|Y) \rangle \) proceeds separately along the same lines, once again however we have to take care of the fact having only available samples from \( M_{\text{obs},j}(X_a|Y_j) \) and not directly from \( \langle P_{\text{obs}}^*(X_a|Y) \rangle \). Similar to 3.25 this can be taken care of by maximizing the weighted log-likelihood using a correspondingly modified EM-procedure. As in the nonparametric case (eq. 3.24), the two thus obtained estimates then permit updating \( f_a \) in the following way:

\[
f_a(x_a|\theta) = \frac{\hat{P}_{\text{obs}}(x_a|\theta_{\text{obs}})}{\hat{M}_{\text{free}}(x_a|\theta_{\text{free}})}
\]

(3.28)

where \( \theta = (\theta_{\text{obs}}, \theta_{\text{free}}) \) is the combined vector of parameter vectors \( \theta_{\text{obs}}, \theta_{\text{free}} \) that control the parameterization of the estimates of \( \hat{P}_{\text{obs}}(x_a|\theta_{\text{obs}}) \) and \( \hat{M}_{\text{free}}(x_a|\theta_{\text{free}}) \) respectively. It can be straightforwardly verified that inserting 3.28 into 3.14 yields an approximation to \( \langle P_{\text{obs}}^*(X_a|Y) \rangle \) as desired. Thereby we obtain an algorithm that consists of two 'nested' EMs: An 'outer' approximate EM that alternates between inference by means of spike-based BP as described previously (quasi E-step) and a subsequent fit of \( P_{\text{free}}(X_a) \) to samples from \( \langle P_{\text{obs}}^*(X_a|Y) \rangle \) (quasi M-step). The fitting procedure of the quasi M-step in turn can then be conducted by an 'inner' EM. The parameterization of \( \hat{P}_{\text{obs}}(x_a|\theta_{\text{obs}}) \) and \( \hat{M}_{\text{free}}(x_a|\theta_{\text{free}}) \) hereby permits proceeding along the lines of Bishop in an analogous fashion (see [15], section 9.2.2).

In this section we have shown different principles for using the output of spike-based BP processors for learning of the factor functions that concretely specify the probabilistic dependencies between variables in a graphical model. Hereby Rolffes approximate EM learning rule [143] served as target for modifications of the individual factor functions. We have shown three approaches for achieving this target, two parametric and one nonparametric approach. Whereas parametric methods define factor functions by a parameterized class of analytic functions, e.g. gaussian mixtures, nonparametric methods are unbiased, as they do not make any assumptions about the general shape of the factor function. However, a potential drawback of the latter might be the inherent roughness of their function representation, when the kernel width is chosen small compared to the sample size. In this case the factor function \( f_a(x_a) \) may change dramatically even for small changes in \( x_a \) and thus induce false dependencies between the components (variables) of vector \( x_a \). As stated in the beginning of this section, we have not yet performed any experiments with the presented approaches and hence so far cannot favor one over the other.
3.3 Functional Description of an Abstract Processor Performing Spike-Based Belief-Propagation in Discrete Graphical Models

Whereas the ISI processor defined in section 3.2 allows for BP in analog graphical models, the abstract processor type laid out in this section is restricted to the presence of discrete variables. However as we will see below, the advantage of the latter is that it makes use of a more general definition of spike-based events, which may even include ensembles of spikes. Hence, this model may provide greater flexibility for its possible concrete implementation in neural wetware or in customized electronic hardware. We will first discuss shortly the general functionality of this processor, which is on a similar description level as the abstract processor of section 3.2. We will then describe a couple of possible event definitions that can be used within this framework. Concrete neural versions will be given in section 5.2. It is important to note however that empirically these implementations were far less successful in general than that of the ISI processor.

As with the ISI processor, we describe the new model for the case of computing output messages in a 3-way factor node that can be generalized in a straightforward manner (fig.3.11a). In contrast to the former model however, so far we have found no way of normalizing these messages and must hence rely on proper scaling of the functions that define factor nodes (see below). When discrete variables are used, these functions can be represented as high-dimensional tables (or tensors), with as many dimensions as variables attached to the associated factor node. Such a tensor thus provides a nonnegative value for each combination of variable states (fig.3.11b). Based on this tensor we can define an event-based processor for SPR computation in the following way (see unnormalized output message $um_{\text{out}}(z_1)$ in fig.3.11c):

The spike trains underlying BP-messages represent stochastic binary processes, such that within a given time window $W$ (grey lines) there can either be some predefined spike-based event (1) or not (0). In a simple case for example, such an event can be the presence of at least a single spike in $W$. However, as we will see below, there are many other possibilities for defining spike-based events. We will refer to the two event-related states of stochastic process $i$ at time $t$ as ‘process states’ $s_{i,t} \in \{0, 1\}$. The discrete states of variables in the graphical model are called ‘variable states’ accordingly. A message $um_X(x)$ consists of dedicated stochastic processes for each variable state $x$. All of these processes are mutually independent. The value of $um_X(x)$ is given by the probability of an event to happen in process $x$ within $W$, i.e. $um_X(x) := P(s_{x,t} = 1)$ (for illustration purposes we here assume all processes to be stationary, i.e. $P(s_{i,t} = 1) = \text{const.} \forall (i,t)$). Time windows $W$ are explicitly defined through an external reference spike train $r$: If there is a spike in $r$ at time $t_w$, then for each input message process it is investigated
whether there is at least one event in $[t_w - W, t_w]$, that is, for each $k \in \{x_1, x_2, y_1, y_2\}$ it is tested if $s_{k,t_w} = 1$. A tensor entry $f(x, y, z)$ is said to be 'activated at time $t_w$' if $s_{x,t_w} = s_{y,t_w} = 1$. In the situation shown in fig. 3.11c for example, the two entries enclosed by the grey box are activated, because $s_{x_1,t_w} = s_{y_1,t_w} = s_{y_2,t_w} = 1$. Activation of an entry means that for a short time period following $t_w$ there is an increased 'firing rate' at the stochastic output process $(s_{z_1,t})$. That is, right after $t_w$ a pulse kernel is added to the instantaneous firing rate of the output, and the integral of that pulse (black striped area) equals the sum of activated entries ($6 + 3 = 9$ in this case). Therefore, the expected number of spikes fired due to the pulse equals the sum of activated entries. We will show now that with the type of processing just described, the expected number of fired output spikes in total is proportional to the SPR. Note however that the stochastic output process generates spikes, whereas we have stated above that the definition of an event can be different from a single spike being present or not. So far it is not clear how the output message $um_{out}(z_1) = P(s_{z_1,t} = 1)$ depends on the expected number of fired spikes and how this transformation distorts the numerical result of the SPR. This issue will be treated shortly.

Proposition:

If the processor of figure 3.11c is implemented for computing $um_{out}(z_1)$, then the expected number of spikes fired by the processor is proportional to $uSPR(z_1)$, i.e. the discretized version of equation 3.1.

Proof:

Let $f(x, y, z_1)$ be the entry of the factor tensor for the variable combination $X = x, Y = y, Z = z_1$, where $z_1$ is fixed. Based on $f$, we can define a binary random variable $f^*_t(x, y, z_1)$ in the following way:

$$f^*_t(x, y, z_1) := \begin{cases} f(x, y, z_1), & \text{if } f(x, y, z_1) \text{ has been activated at time } t \\ 0, & \text{otherwise} \end{cases}$$

We will show now that the expected value of quantity $F^*_t(z_1) := \sum_{x \in \{x_1, x_2\}} \sum_{y \in \{y_1, y_2\}} f^*_t(x, y, z_1)$ is equal to the SPR, and that the processors output follows $F^*_t(z_1)$.

Each spike of the reference input at some time $t_w$ can be interpreted as a sample drawing of $s_{x,t_w}$ and $s_{y,t_w}$, at each stochastic message input $x \in \{x_1, x_2\}, y \in \{y_1, y_2\}$. This then automatically leads to a sample drawing of each $f^*_{t_w}(x, y, z_1)$. Since the stochastic processes underlying the individual messages are mutually independent, the probability $p(x, y)$ of the tensor entry $f(x, y, z_1)$ to be activated after a
sample drawing is \( p(x, y) = um_X(x) \cdot um_Y(y) \). Hence the expected value of \( f_{t_w}^*(x, y, z_1) \) is given by:

\[
E[f_{t_w}^*(x, y, z_1)] = f(x, y, z_1) \cdot um_X(x) \cdot um_Y(y)
\]

from the linearity of the expected value operation it then follows that the expected value of \( F_{t_w}^*(z_1) \) is given by:

\[
E[F_{t_w}^*(z_1)] = \sum_{x \in \{x_1, x_2\}} \sum_{y \in \{y_1, y_2\}} E[f_{t_w}^*(x, y, z_1)] = \sum_{x \in \{x_1, x_2\}} \sum_{y \in \{y_1, y_2\}} f(x, y, z_1) \cdot um_X(x) \cdot um_Y(y) \tag{3.29}
\]

which is the discrete analogon to SPR equation\(^4\) in case of unnormalized input messages \( um_X(x), um_Y(y) \).

It remains to be shown that the expected number of output spikes fired by the processor is indeed proportional to \( E[F^*(z_1)] \). After each spike at the reference input, a firing rate pulse \( r^*(t - t_w) := F_{t_w}^*(z_1) \cdot r_0(t - t_w) \) is generated at the output and added to the rate curve \( r_0(t - t_w) \) denotes the stereotyped template waveform of the pulse with unit area). The summed pulses yield a doubly stochastic process at the output, because not only the individual spikes but also their underlying instantaneous rate profile thus becomes a random quantity. The integral of a single pulse \( r^* \) over \( t \) is the expected number of spikes fired due to \( r^* \) and is given by the sum of activated tensor entries \( F_{t_w}^*(z_1) \). Therefore, after \( n_{\text{ref}} \) reference events (sample drawings) at times \( \{t_1, ..., t_n_{\text{ref}}\} \), the expected area \( n_{z_1} \) under the instantaneous rate curve is given by:

\[
E[n_{z_1}] = E \left[ \int_{-\infty}^{\infty} \sum_{i=1}^{n_{\text{ref}}} r^*(t - t_i, t_1) \, dt \right] = E \left[ \sum_{i=1}^{n_{\text{ref}}} F_{t_w}^*(z_1) \right] = n_{\text{ref}} \cdot E[F^*(z_1)] \tag{3.30}
\]

which of course is also equal to the expected number of fired spikes. \( \square \)

We are now in a situation where the output rate of spikes \( r_{z_1} \propto E[n_{z_1}] \) follows the SPR of the input messages (probabilities) \( um_X(x), um_Y(y) \). However, since the output message \( um_{out}(z_1) \) might be used as input to a subsequent factor, we have to achieve consistency between the encodings of \( um_X(x), um_Y(y) \) and \( um_{out}(z_1) \). This directly leads to the question of how \( um_{out}(z_1) = P(s_{z_1,t} = 1) \) depends on \( r_{z_1} \). This relationship of course is different for each definition of spike-based events. In the ideal case the relation is linear, but even in the absence of linearity the methods given in \( \text{[63]} \) can be used to estimate the inference error associated with the thus induced message errors. In the following section we provide some exemplary definitions of spike-based events, which lead to output messages that are equal to the SPR after application of some squashing function. A further unresolved issue is normalization: Since the event probabilities \( P(s_{z_1,t} = 1) \) are independent for different values of \( z_1 \), there is no implicit normalization
of $\text{um}_{\text{out}}(z_1)$ and it is unclear how it can be achieved explicitly by the processors functionality. Again, among other things the severity of this issue depends on the definition of spike-based events, however, for the first of the following examples we can provide an easy fix.

### 3.3.1 Examples Of Spike-Based Events

A straightforward way to define events in spike trains of rate $r$ is the following:

>'An event has happened if there had been at least $n > 0$ spikes within time window $W$'

Assuming Poisson statistics, the probability $P$ of such an event to happen is given by

\[
P(n | r, W) = 1 - \sum_{i=0}^{n-1} \frac{(rW)^i}{i!} \cdot \exp[-rW]
\]  

(3.31)

If, as before, rate $r$ is proportional to $u_{\text{SPR}}$ it is apparent that this formula 'squashes' the SPR outcome through its sigmoidal shape as a function of $r$. In general the overall error of the inferred marginals due to such squashing depends on the whole graph and the algorithm provided in [63] permits its assessment in this case. However, a first means for diminishing the message error is to uniformly scale down the entries in the factor table. From [2] it is clear that this leaves unaffected the factorization represented by the graph, but at the same time it restricts the output message to the linear part of squashing function since downscaled factor table entries automatically lead to downscaled firing rates (see eq.3.30 and 3.29). The price to pay is a larger number $n_{\text{ref}}$ of sample drawings to correctly assess the true messages $\text{um}_{\text{out}}(z_1)$. In section 5.2 we will have a more detailed look at the squashing-induced message error, when a concrete neural implementation for the case $n = 1$ will be given.

Another possibility for defining a spike-based event is to demand that:

>'within $W$ there has to be at least a single burst.'

where a burst can, for example, be defined by the occurrence of two successive interspike intervals smaller than some predefined constant. The resulting squashing function will then depend on various details of the processor, e.g. the reference spike process and the width of the rate pulse $R(t - t_w, t_w)$. In general there is an abundance of possibilities of how an event can be defined with spikes, e.g. one might also think about specific temporal spike patterns. What needs to be fulfilled for the approach to remain valid however, is the condition that the sequence of events follows a renewal process (but not necessarily a Poisson process). For example one might think of a spike train created by adding specific spike patterns together in an i.i.d fashion: For each addition, a pattern is randomly drawn out of some predefined set and 'pasted in' after the previous one. The problem then is however to construct a spiking circuit whose output can reproduce the same spike patterns -since this output might be used as input to
a subsequent processing stage. The difficulty of finding such a circuit is also the reason why the output in fig. 3.11 produces spikes rather than general spike-based events.

Figure 3.11: a) Factor node attached to three discrete variables (edges $X, Y, Z$). Colored arrows stand for Belief-Propagation messages arriving at the node (red/blue), or being sent away to neighboring nodes (green). Each message provides a probability for each state of the variable it is passed along.

b) Three dimensional tensor implementing a specific instance of the factor node in (a). Variables $X$ and $Y$ each have two states ($x \in \{x_1, x_2\}, y \in \{y_1, y_2\}$), whereas $Z$ has three states ($z \in \{z_1, z_2, z_3\}$). Each combination of variable states is associated with a nonnegative value by the tensor.

c) Abstract processing of the SPR output message along $Z$ encoded by spikes. Illustrated is the processing of the message value $um_{out}(z_1)$ of the first state of $Z$ (= first page of the tensor in (b) ). The input messages $um_X(\cdot)$ and $um_Y(\cdot)$ are also encoded by spikes (not shown), such that within some time window $W$ ending at time $t_w$ there can either be a predefined spike-based event (1) or not (0). See text for details.
3.4 Chapter Summary & Discussion

We have presented in this chapter two abstract approaches for spike-based belief-propagation (BP) that have in common the same functional meta-principle. The key observation underlying this principle is the mathematical equivalence between the sum-product rule (SPR) and the general formula for computing expected values of functions of random variables. From this follows the possibility to implicitly compute the SPR through basic Monte-Carlo (MC) techniques, i.e. by drawing random instantiations of these variables, applying the factor function to them and sum the result. The central limit theorem then assures the asymptotic correctness of such a sample-based representation of BP messages, in the limit of a large number of sample drawings. This principle offers the advantage that, in any concrete implementation, no specialized functional blocks are needed for the individual arithmetic components that constitute the SPR (sums\integrals and products). It is the application of MC methods that allows the required computation to happen implicitly, in a similar way as the dynamics of systems in nature might be considered to be the result of complex computations on natures’ mathematically formulated laws. This also naturally solves the problem associated with the intractable integrals that occur when the SPR is applied to analog graphical models.

Whereas the first presented processor implementing the meta-principle is tied to the distinct interpretation of a single spike as a random sample of the underlying interspike interval (ISI) distribution, our second presented approach is more flexible, in that it allows for a multitude of spike-based definitions of random samples. So far however we have not yet examined this second processor in practice, the present chapter has dealt only with its general functionality. Concrete neural implementations will be considered in section 5.2.

On the other hand, the ISI processor is representationally more powerful, since it can solve inference problems in discrete as well as analog graphical models. We have shown this analog processing ability for a single equality-constraint factor, that is, an important type of factor that endows Forney Factor Graphs (FFGs) with the full representational power of ordinary Factor Graphs. Moreover, for two distinct graph structures, it has been verified that an ensemble of interconnected processors is able to perform inference in randomly parameterized, analog FFGs. The correctness of this method in the discrete case follows straightforwardly, since discrete factor nodes can always be emulated as piecewise constant analog factors. The proposed method has not only been applied to static inference tasks, but also to a simple HMM that involved inference over time. Finally we have shown how the sample-based representation of BP messages can be utilized by an Expectation-Maximization (EM) method from the literature for learning the parameterization of factor nodes. This links our inference approach to the biologically highly relevant problem of unsupervised concept learning from data.
The ability to process analog variables poses a clear advantage of such neurally inspired computational principles over traditional methods in computer science, where the difficulties associated with BP in analog graphical models have long been recognized [59, 101, 90]. Although in the latter case sampling approaches have also been applied [64, 161, 29], the intrinsically analog nature of neural elements (synapses, somas, dendrites etc.) suggests that these components in general pose a more suitable environment for analog processing compared to digital computers. Sampling methods in particular could be a natural basis of neural computation, given the wealth of data showing large trial to trial spike variability in experiments [116, 164, 155, 61, 151, 82]. Biophysically such noisy behavior is thought to either stem from the large number of uncorrelated, balanced inhibitory and excitatory synaptic inputs of a real neuron [152, 169, 8, 55, 80, 8], or from the necessarily finite number of stochastic ion channels in the neurons cell membrane [178, 145, 40, 146]. Later chapters will deal in slightly more detail with noise in simplified neuron models. In the present chapter however we have made no conjecture about the underlying cause of noise in neurons, it has rather been the noise’ effect on spike output and its potential role in neural computation that has been of interest.

In particular, the hypothesis that this role is embodied in the ISIs of renewal spike trains has to be verified experimentally. Indeed, the surprisingly few studies that have examined renewal properties of cortical spike trains have reported only low correlation coefficients around $-0.2$ between two adjacent ISIs (in cat [179] and rat [115] layer 5 somatosensory cortex). Also, at least in rat, correlations between an ISI and its $k$-th nearest neighboring ISI ($k > 1$) were virtually zero. Future work has to verify if the proposed ISI processor can still work in presence of such slight deviations from the renewal assumption. Apart from that, a substantial number of studies has demonstrated the significance of ISI-based processing in a variety of low-level sensory systems across species, sensory modalities and brain areas [167, 25, 166, 138, 39, 97, 137, 153]. For example, in the motion-sensitive neuron H1 of the fly visual system, changes in stimulus variance can be unambiguously detected using ISI histograms –in marked contrast to information solely provided by instantaneous firing rates [97, 39]. This detection happens very fast, allowing perfect stimulus discrimination after $\sim 8 - 12$ ISIs. In macaque visual area V1 [138] and cat auditory area A1 [153], spikes preceded by short ISI lengths contain a higher amount of information about the stimulus and show greater feature selectivity [153]. Also, the efficacy of spikes in cat retinothalamic [167] as well as thalamocortical axons [166] for driving a spike in respective postsynaptic neurons smoothly depends on the spikes preceding ISI. Finally on the psychophysical level, a large variety of features of ISI distributions in the auditory nerve and cochlear nucleus can be mapped directly to features of pitch perception [25]. The ISI encoded stimulus information in low-level areas, e.g. primary sensory cortices, must be picked up by neurons in higher-level areas, e.g. secondary sensory and association cortices, in order to infer the identity of the stimulus and/or to select appropriate commands -as in motor tasks for example. Hence,
the sparsest assumption is an uniform coding scheme, such that neurons in high-level areas preserve the ISI coding of neurons in low-level areas. Many studies however have reported consistency of in vivo ISI distributions of nonbursting cells with a class of unimodal distributions (quasi-exponential, gamma and log-normal distributions) in area MT [155 10], posterior parietal and dorsolateral prefrontal cortex [26] and suprasylvian gyrus [116 23]. These results are usually taken as indications for quasi-Poissonian firing and so point to a rather restricted class of ISI distributions present in higher-level areas of cortex. Figure 3.7a shows a typical experimental ISI histogram on which this claim is based. The low diversity of ISI distributions in turn seems to be inconsistent with our abstract processor, which is expected to produce arbitrarily shaped BP messages.

However, by performing sampling based on the log(ISI)- rather than the ISI-label of a spike, we have shown that the output of the abstract processor can be qualitatively reconciled with the above class of distributions. Moreover, relying on logarithmic quantities permits the representation of variables with a negative domain and might also be advantageous for computations where variables with drastically different dynamic ranges are combined (for example in a factor node relating stimulus information from different sensory modalities).

A further way of reconciling our processor with the experimental data is based on the long electrode recording times in experiments. Typically spike data recorded over the course of seconds is pooled to produce an ISI histogram [155 10]. On the other hand, BP messages in Factor Graphs containing cycles are generally not stationary from the beginning of an inference trial - in fact message passing may not converge at all [112]. Therefore, if experimental recording takes place in a loopy structure, then each recorded spike corresponds to a snapshot message during the convergence process, which in turn may lead to smooth empirical ISI distributions that reflect the average of messages computed during the whole inference trial, rather than the final steady-state messages only. This may explain the lack of features for large ISI values in some experimental distributions - and the corresponding quasi-exponential appearance of the latter - because, if messages are encoded by log(ISI) distributions, these features are more susceptible for being averaged out compared to the consistent peak at small ISI values.

Apart from that, the hypothesis of Poissonian firing itself may be put into question. Indeed, the outcome of some experimental studies is quite inconsistent with the Poisson assumption [104 138 153]. In [104] the authors have shown that spike time regularity, i.e. unimodal sharpness of ISI distributions assessed by the coefficient of variation, is larger in high-level association and motor-like parietal regions (area LIP and area 5 respectively, see fig. 3.7b), than in areas lower in the hierarchy (areas MT and MST, fig. 3.7a). If cortex performs computations based on Factor Graphs, unambiguous inference results are expected in high-level areas, where stable interpretations of the world are formed and reliable motor commands have to be read out for execution. In our model, unambiguous marginal probabilities correspond
exactly to highly regular spike trains. Moreover, also primary sensory areas are found to be inconsistent with the Poisson assumption, if histograms of the log(ISI) values rather than the ISI values are formed (see figure 3.7) and also [39] for log(ISI) data from the fly). This finding holds true although the corresponding ISI distributions appear quasi-exponential (see [138], figure 1). Log-ISI histograms however make features apparent that are barely visible in linearly scaled histograms [133].

Besides ISI coding there are several other models using various spike-coding schemes, that propose how Belief-Propagation [135, 120] or other methods of statistical inference [181, 98, 17, 98, 87] could be performed by spiking neurons. However, these models are limited in that they are confined to a restricted set of graphical models (tree-shaped Bayesian networks [32], naive Bayesian networks [181, 98, 17]), or depend on rough biophysical approximations [135]. Furthermore, except [98, 17] all other approaches are based on binary [120, 32] or general discrete variables [181, 135, 156, 87]. In contrast, our model is very general, in that it allows inference in Forney Factor Graphs containing analog variables. As mentioned before, FFGs belong to the most general types of graphical models, since they subsume Bayesian Networks (as in [181, 98, 135, 32, 17]) as well as Markov Random Fields (as in [120]). Bayesian Networks and Markov Random Fields however are restricted in so far as there are instances of graphs of either type which cannot be translated into the corresponding other one, while keeping the same set of variables [15]. This advantage however comes for a price: In contrast to other authors work [181, 98, 32, 17] we can provide only a less detailed experimental evidence supporting our model. Our focus has mainly been put on algorithmic generality, rather than mimicking the results of psychophysical studies in specific brain areas. The generality of FFGs allows Bayesian inference to be applied in a broader context, e.g. to motor control and face recognition problems [13, 161], whose corresponding graphical model structures exceed the naive Bayesian networks expressed in pure optimal cue combination. For example think of a human baby trying to imitate the vocal pronunciation of its parents (see [13] last chapter). When observing the parents’ lips, while hearing their distinct vocal sound (cue combination), the baby must select appropriate commands for its own lip and tongue muscles in order to properly imitate the vocal (motor action selection).

Given the generality of results presented in section 3.2.2 we have no reason to assume that the abstract processor cannot be easily applied to such cognitive modeling. On the contrary an electronic hardware version for example might even be beneficial, because it abolishes the need for cumbersome discretizations that accompanies software realizations (e.g. as in [13] last chapter).
Chapter 4

Results II: Implementing Belief-Propagation in Concrete Spiking Neural Networks Based on Liquid-State Machines

In contrast to the approaches presented in chapter 3, we here discuss a model of spike-based Belief-Propagation (BP) that is realized concretely by software-simulated neural elements. That is, this model consists not just of a mathematical description of some spike-based processor, rather its functionality is implemented directly by the simplified biophysical properties of the spiking neuron models discussed in section 2.3. Such implementations of predefined functionalities are feasible, since the presented approach is based on the Liquid-State Machine (LSM) framework, which allows for approximations of general, time-invariant filters with fading memory (see section 2.4). We first present a general LSM-based architecture that allows for neural implementations of general message-passing algorithms. Hence, this material may have a broader impact beyond the BP framework. Using this architecture we then concretely implement BP in Forney Factor Graphs (FFGs) with binary variables. Because the spike-coding scheme used with this version of BP cannot be generalized straightforwardly to FFGs with analog variables, concrete neural implementations of the latter are only presented in chapter 5.
4.1 A General Model for Implementations of Message-Passing Algorithms in Neural Populations

As stated in the introduction, the appeal of message-passing algorithms for modeling biological neural networks in general relies not only on their innate processing of exclusively local information, there is also an intriguing similarity between synaptic bouton structure in cortex and graphs in particular, as exemplified in fig.1.1. This finding has motivated our development of a LSM-based architecture, that allows general message-passing algorithms to be executed in spiking neural networks and which is presented here.

Figure 4.1 illustrates the basic principle behind this approach. Any given message-passing algorithm playing out on some graph structure (here the connectivity of the nodes \(f_1, \ldots, f_4\)) can in principle be implemented by an interconnected arrangement of LSMs in the shown fashion. A message send by some node \(f_a\) to a neighboring node \(f_b\) is computed by a combination of a reservoir (liquid pool \(L\)) and a linear readout device \((R)\). Depending on the context, the readouts may either consist of individual neurons or whole populations. Since the reservoir serves as a general-purpose filter bank (see section 2.4), rather than having a dedicated reservoir for each individual message, a sparser solution is to use just one per node in the graph. This is a reasonable architecture for message-passing algorithms, because due to the local information processing of the latter, it is assured that each message emitted by some node only depends on the input messages that node receives and not on any other source of information. Furthermore, additional readouts that do not participate in the message-passing process per se, but which compute quantities of interest that depend on some subset of input messages, might also be attached to a single reservoir. If multiple reservoirs per node were used, each of them would independently constitute a bank of filters that were all dependent on the same set of inputs. Hence, such an arrangement would be highly redundant. However, in case of BP this argument is not strictly true, because each output message is supposed to ignore the input message that comes in along the same edge (see eq. 2.9). In this case having a dedicated reservoir for each output message would generally facilitate the learning of the readout, since, for each reservoir, the superfluous message could be intentionally prevented from providing input to the reservoir. The larger the degree of the node however, the less beneficial the vast number of additional reservoirs will be. In general there is a tradeoff between the amount of resources to be spent for the realization of learning machines and the approximation quality of the target function to be learned. As we will see in the next section, with the type of spike-encoding of messages used there, even for factor nodes of degree three, the targets can be approximated reasonably well by the architecture of fig.4.1. However, the same does not hold true for the concrete implementation of the abstract spike-based BP processor in section 5.1. The results presented there are indeed based on the presence of a dedicated
reservoir for each message. Further empirical research on the basis of simulations is needed to assess whether the architecture of fig. 4.1 can overcome this limitation.

Further research is also needed with respect to the control inputs $C_i$ in fig. 4.1. Their existence potentially greatly enhances the computational power of the interconnected LSM arrangement. For example, they might be used to switch entirely the computational task performed by the arrangement. Indeed Maass et al. [99] have shown that in a LSM with readout feedback (see section 2.4), pulse-like control signals can be used to switch the task performed by the LSM from e.g. subtraction to addition of two inputs. Transferred to the architecture of fig. 4.1 an abundance of possibilities occur. For example, the LSM arrangement might switch its task from computing marginal probabilities by means of BP, to computing the 'maximum a posteriori' (MAP) estimate [15, 76] by means of the 'Belief-Revision' / 'Max-Product' algorithm [125, 131] (which is some different message-passing algorithm). Alternatively the inputs $C_i$ might be used to modify the parameterization of the local factor function $f_i$. Depending on the ensemble of $C_i$s, the graph as a whole may then represent different dependencies between potentially different variables. Thereby one can use the same neural machinery for solving a variety of mutually unrelated inference problems. Another possibility for using the control inputs would be to embed multiple distinct graph structures within a single, potentially highly cyclic graph. The controls may then be used to selectively switch off those nodes or edges that are absent in a particularly selected graph. Such silencing of connections is an essential feature, for example, of the tree-based reparameterization framework for inference that generalizes BP [171].

The results in this thesis however have been obtained in the absence of the controls $C_i$. As stated above, further empirical research is needed to evaluate their utility.

Figure 4.1: General architecture for embedding message-passing algorithms in spiking neural networks. Each node $f_i$ of the graph on which message-passing plays out corresponds to a single reservoir (liquid pool ($L$)). Messages send by a node $f_a$ to $f_b$ are produced by dedicated readout pools ($R$) that receive input synapses from the reservoir corresponding to $f_a$ and project output synapses to the reservoir of $f_b$. Control inputs $C_i$ are optional and may serve several different purposes. (see main text for details)
4.2 Encoding Belief-Propagation Messages with Population Rate Codes

With the arrangement of LSMs of the previous section in mind, we are now ready to look at concrete implementations of BP in simulated spiking neural networks. The core models of the neuron and synapse types used here, as well as basics of LSMs, are explained in the background (section 2.3 and 2.4 respectively). Although these models are still crude abstractions of real, biological neurons, they do capture some core principles of the latter. We begin this section by laying out the specific BP message update equation that underlies the presented approach. Using population rate codes, we then explain how such messages can be represented by spikes and how LSMs can be trained to approximate them. Consequently results of spike-based message computations by a single LSM node are presented thereafter. The section ends with evaluations of interconnected ensembles of LSMs that solve inference problems in full FFGs.

4.2.1 Message Update Equation, Population Rate Coding And Training of The Liquid-State Machines

4.2.1.1 Message Update Equation

Compared to equations 2.9-2.10 the BP message update equations used here differ in two aspects. Firstly, because discrete variables are used, all integrals are replaced by corresponding sums over the domain of these variables. For simplicity, we focus on binary variables, although in principle the approach is applicable also to variables of arbitrary base. Secondly, unlike eq. 2.9-2.10 the message updates are not instantaneous, rather they follow first-order low-pass dynamics. In addition, some delay is associated with these updates. Such delays are necessary to respect the inevitable transmission delay of synaptic communication (see section 2.3). Low-passing in turn permits accommodating for the time-integrative properties that accompany reservoir approaches. Moreover, in practice it is often helpful for the BP algorithm to converge in cyclic graphs if the messages are only partially updated at each time step [180], and low-passing can be regarded as the extreme case of an infinitesimal update \( dm \) during 'time step' \( dt \). Whereas, in the abstract processor approach of section 3.2, the use of time window \( W \) also implicitly leads to low-pass filtering, here we make this filtering explicit in the dynamical systems the LSMs have to emulate.

In case of a binary variable \( X_{ij} \in \{0, 1\} \) a (normalized) message sent by node \( i \) to node \( j \) at time \( t \) may consist of only a single scalar value \( m_{i \rightarrow j}(t) \) for state \( X_{ij} = 1 \). The message value for \( X_{ij} = 1 \) is then implicitly provided through normalization. Taking the above two aspects into account the dynamical
The equation for \( m_{i \rightarrow j}(t) \) becomes:

\[
\tau \dot{m}_{i \rightarrow j}(t) + m_{i \rightarrow j}(t) = \frac{1}{N_{i \rightarrow j}(t)} \sum_{x \in \{0, 1\}^n} f_{i}(x) f_{i}(x) \prod_{k \in \mathcal{N}(i) \setminus \{j\}} p_{k \rightarrow i}(x_{ik}, t - D)
\]

where \( m_{i \rightarrow j}(t) \) is the scalar message passed from node \( i \) to node \( j \) at time \( t \), \( \dot{m}_{i \rightarrow j}(t) := \frac{d}{dt} m_{i \rightarrow j}(t) \); \( \tau \), an arbitrary time constant determining the overall speed of the message updates; \( \mathcal{N}(i) \setminus \{j\} \) indicates the set of neighbors of node \( i \) besides \( j \); \( n \) the cardinality of \( \mathcal{N}(i) \setminus \{j\} \); \( \mathbf{X} = (X_{ik})_{k \in \mathcal{N}(i) \setminus \{j\}} \) is the vector of binary variables \( X_{ik} \) each of which links nodes \( i \) and \( k \); \( f_{i} \) is the factor associated with \( i \); \( D \) is a fixed transmission delay (synaptic delay); \( p_{k \rightarrow i}(x_{ik}, t) = \{ m_{k \rightarrow i}(t), \text{if } x_{ik} = 1; 1 - m_{k \rightarrow i}(t), \text{otherwise} \} \); and \( N_{i \rightarrow j}(t) \) is a normalization term assuring that \( 0 \leq m_{i \rightarrow j}(t) \leq 1 \). \( N_{i \rightarrow j}(t) \) therefore depends on the \( f_{i}(x_{ij} = 1, \mathbf{X}) \) as well as on the \( f_{i}(x_{ij} = 0, \mathbf{X}) \).

We now describe a method for encoding \( m_{i \rightarrow j}(t) \) with spikes. This method forms the basis of the simulation results presented in sections 4.2.2 and 4.2.3.

4.2.1.2 Population Rate Coding of BP Messages

We use the population activity/rate\footnote{\cit{55}} of readout populations as a method for encoding time-series of the messages \( m_{i \rightarrow j}(t) \) in equation 4.1 with spikes. Therefore, the spiking neurons of the readout pools (figure 4.1) should encode messages \( m_{i \rightarrow j}(t) \) (probability values) using a population rate code that depends linearly on the encoded probability value.

The problem is then, how to arrange for the readout neurons to report the correct output message in response to the current state of the liquid pool -which depends on the time-series of the input messages? In summary, our solution is to drive the liquid with a broad range of possible input time-series (spike trains), and to train the readout pools to generate their corresponding output messages. When suitably trained, these pools will be able to generalize their learned behavior to generate appropriate output messages in response to novel input messages injected into the liquid. Hence we expect the readouts to still compute the correct output, once the externally applied input is replaced by the readout output from some neighboring LSM, as it is the case for fully interconnected FFGs (figure 4.1).

If a general filter of the kind in equation 4.1 is to be implemented by a LSM, the training examples must cover a wide range of possible input time-series \( m_{k \rightarrow i}(t) \), because it is not known in advance what time-series some liquid pool \( i \) will be exposed to when operated in the fully connected scenario. Therefore, training examples are generated as follows: Low-pass filtered white noise (an Ornstein-Uhlenbeck process with reflective bounds) is used to generate random population rate curves, each coding for a possible message signal \( m_{k \rightarrow i}(t) \). These patterns represent the instantaneous firing rate underlying an inhomogeneous Poisson spike process. Since each rate curve determines a population message signal, the
same curve determines the instantaneous rate of each neuron in an input population. The resulting spike
trains had a duration of 120-300s in simulated biological time.

We look at two versions of population codes, each of which has a different linear relation to the
encoded BP message. In the first case, the fraction of active (firing) neurons within a sufficiently small
time window directly codes for the scalar probability of a message (see eq. 4.1). This is the coding scheme
of the p-Delta learning rule [9, 101] that is used for training the readouts. The p-Delta rule however is
formulated for the McCulloch-Pitts (Perceptron) neuron model, that neither incorporates a reset of the
membrane potential nor a refractory period. As we will see later, these aspects cause problems when the
weight vector learned by the p-Delta rule is used to drive a physiologically more realistic IAF neuron.

To overcome this difficulty, the second type of population code operates directly on IAF neurons. Its
linear relationship with message values is such that probability values 0/1 respectively correspond to a
population rate of 0/90Hz. In order to generate the spiking readout messages of our choice, we have first
to determine the relationship between the total mean input current \(I\) provided by the synapses from the
liquid neurons to the readout neurons, and the population rate \(R\) of the latter. This step is necessary
to ensure consistency between the encoding of the (training) input messages and the readout response
which, after training, provides the input to a neighboring liquid pool. We can obtain this relationship
empirically by conducting several simulations with independent samples of networks of the following type
(the parameter values in brackets are those that underlie the results presented in sections 4.2.2 and 4.2.3):

All neurons \((N = 343)\) belonging to the same readout population are mutually unconnected and their
threshold values are drawn independently from a uniform distribution. Each neuron receives a diffusive
(white) noise current, however all of them receive the same temporally varying mean input current \(I\) (see
[154, 21] for similar networks, but without distributed threshold values).

One can then perform a set of trials (20), in each of which a random instance of such a network is
created. A random mean current trace \(I(t)\) is subsequently injected into all the neurons. This current,
together with additive gaussian noise, then evokes Poissonian spike trains of varying instantaneous firing
rates in the neurons. All neurons receive the same mean current. However, their individual white noise
processes are independent. The resulting population rate \(R(t)\) is due to the combined firing of all neurons
and can be computed as follows: The spike trains of all neurons in a given trial are merged into a single
one and \(R(t) = \frac{1}{N} \frac{dC}{dt}\) is determined, where \(C(t)\) is the (smoothed) total number of spikes fired until
time \(t\) by the whole population. The \(I(t)\) and \(R(t)\) data of all trials are lumped together and the \(I(R)\)
relationship is determined by polynomial fitting. This relationship is then used to determine the desired
target current during the supervised learning procedures of the readouts. We use a linear dependency
between \(R\) and the message \(m \in [0, 1]\) encoded by it: \(R = m \times 90\) Hz.

After learning, the current \(I\) is provided by the sum of (trained) postsynaptic currents (PSCs) injected
by the neurons of the presynaptic liquid pool. The distributed firing thresholds in the readout populations have an important role in that they permit the population responses to depend mainly on the actual \( I \) and less on past values of this current [M.Bethge personal communication]. The goal is to map instantaneously \( I \) together with the noise onto a corresponding population rate without introducing dependencies on the distant past, which the liquid pool is unable to provide because of its fading memory [101].

### 4.2.1.3 Setup And Training of The Liquid-State Machines

The spiking neuron simulations in the next section are implemented using the software package CSIM (available at the Institute of Theoretical Compute Science/TU-Graz, [www.lsm.tugraz.at](http://www.lsm.tugraz.at)). The organization of our liquid pools follows closely that of Maass et al. [101]. We use leaky-integrate-and-fire (LIF) neurons (20% inhibitory) located on the integer points of cuboid lattices with a distance dependent gaussian connection probability. A synaptic connection between two neurons with position vectors \( \mathbf{a} \) and \( \mathbf{b} \) is established with probability \( p = C \cdot \exp \left( -\frac{\|\mathbf{a} - \mathbf{b}\|^2}{\lambda^2} \right) \). We use only current injecting synapses, which were modeled as low-pass filters of the input spike trains (see section 2.3).

Spike inputs (SI) to a liquid pool are provided either externally by a population of 'dummy' neurons (for modeling prior knowledge or observed variables); or by a readout pool (\( R \)) consisting of a population of spiking LIF neurons (used for messages computed within the network) (see figures 4.4, 4.5, 4.6 and 4.7). Input neurons of both types connect to neurons of the postsynaptic pool with a probability that is independent of distance. Detailed values of all involved parameters can be found in the appendix section C.1.

For the results presented in the next section, neurons in liquid pools did not receive any diffusive noise current (in contrast to readout neurons in case of noisy readout populations). However, the performance is robust even in the presence of noise in the liquid pool (see appendix C.2).

Using the measured \( I(R) \) relationship, the goal of learning is to determine the synaptic weights from the neurons in the liquid-pool to the readout neurons, such that the latter population fires with the desired rate. For that, the first step is to record the response spikes of the liquid neurons for each training input message. These spikes are then convolved with an exponentially decaying kernel to obtain the unweighted PSC curve for each synapse. This procedure mimics the effect of the current-based synapses described in section 2.3. The resulting curves form the input data of the learning procedure used to determine the synaptic weights.

In case of the p-Delta rule, a suitable target for learning is the evaluation of \( m_{i \rightarrow j}(t) \) in equation 4.1 for each of the input messages. This is sufficient because \( m_{i \rightarrow j}(t) \) may correspond directly to the target fraction of active perceptrons at time \( t \).

In contrast, when a population of IAF neurons is to be learned as readouts, one needs to compute the
target total synaptic current as follows: Firstly, the desired population rate to instantiate the required output message is calculated from equation 4.1 and the linear dependency between encoded message (probability) \( m_{i \rightarrow j}(t) \) and population rate. Then the desired input current is derived from the empirically measured \( I(R) \) relationship. The \( I(R) \) relationship assures that this current (together with noise) will result in the correct population rate encoding of the output message \( m_{i \rightarrow j}(t) \).

For computational simplicity we use linear regression to adjust the synaptic weights towards the desired input current, however in future work this task is expected to be solved within the framework of reward-modulated STDP learning [85]. In the following simulations the synaptic weights are the same for all neurons in the same readout population. Consequently, after successful learning all the readout neurons receive the same mean synaptic current through the synapses from their presynaptic liquid pool. All readout populations are trained separately, and then connected to their presynaptic liquid pool.

4.2.2 Computation of Single Messages

The previously described methods allow for an appropriate setup of LSMs and for an assessment of the performance of single message computations. We first present corresponding data obtained by application of the p-Delta rule. The more successful results using noisy readout populations are shown subsequently.

Figure 4.2 shows the results of a LSM trained to approximate the message output \( m_O(t) \) of a parity-check constraint factor as in [90] (see section 4.2.3 for definition of this factor). Applying template equation 4.1 to such a factor yields:

\[
\begin{align*}
t' &:= t - D \\
\tau m_O(t) + m_O(t) &= \frac{(1 - m_{I1}(t'))m_{I2}(t') + m_{I1}(t')(1 - m_{I2}(t'))}{m_{I1}(t')m_{I2}(t') + (1 - m_{I1}(t'))m_{I2}(t') + m_{I1}(t')(1 - m_{I2}(t')) + (1 - m_{I1}(t'))(1 - m_{I2}(t'))}
\end{align*}
\]

(4.2)

It is apparent from fig 4.2b,c that application of the p-Delta rule leads to an approximation of the target output message defined by this equation. That is, after learning the fraction of active perceptrons reasonably follows \( m_O(t) \). It can also be seen that translation of the perceptron weights into corresponding synaptic weights of IAF neurons is associated with an error, that cannot be attributed to error in learning (compare black and magenta lines). As stated above, this additional error is due to the perceptrons neglect of voltage reset and refractory periods. Indeed the perceptrons in (b) and (c) show extended periods of constant activity (third row), a feature more biologically realistic IAF neurons cannot provide due to voltage reset and the refractory period.

A concomitant experiment reveals that these problems can be overcome when a population of noisy IAF neurons is considered, rather than a set of perceptrons. Figure 4.3 shows the corresponding results.
In this case an equality constraint factor node is simulated, which, upon application of eq. 4.1, leads to the following dynamical system:

\[ t' := t - D \]

\[ \tau \dot{m}_O(t) + m_O(t) = \frac{m_{I1}(t')m_{I2}(t')}{m_{I1}(t')m_{I2}(t') + (1 - m_{I1}(t'))(1 - m_{I2}(t'))} \]  

(4.3)

Clearly in this case the correlation between target and actual output time series is higher than for the p-Delta rule ($CC \approx 0.91$ vs. $CC \approx 0.76$). Although this comparison is slightly unfair due to the different dynamical systems being simulated, its results hold true in general.

In the following section, we will therefore put an emphasis on those whole-graph LSM simulations which are based on noisy readout populations.
Figure 4.2: Computation of a single message by a LSM trained by the p-Delta rule
a) Schematic of the emulated parity-check factor node, together with input messages \(\{m_{I1}(t), m_{I2}(t), m_{I3}(t)\}\) and output message \(m_O(t)\).
b) Top row: Spike trains of the three input populations (representing \(\{m_{I1}(t), m_{I2}(t), m_{I3}(t)\}\)) for testing data not seen during training. 17 neurons are coding each message, y-axis: neuron index. Each black dot marks a spike. The red curves give the sinusoidal instantaneous firing rate used to create the spike trains. Phase, Amplitude and frequency of the sinusoids are random. Second row: Target output \(m_O(t)\) (green line) together with 300 perceptron readout output (= fraction of active perceptrons, black line). Correlation coefficient (CC) and mean absolute error (MAE) between green and black line are 0.755 and 0.103 respectively. The magenta line is the output message decoded from the rate of an IAF-population with the same weights as the perceptrons. Third row: Output of the perceptron population underlying the black line in the second row. y-axis: Perceptron index. Each small dot marks a time-point when the corresponding perceptron is active (= above threshold). Bottom row: Spike output of the IAF population underlying the magenta line in the second row.
c) The same as in (b) for input messages created by an Ornstein-Uhlenbeck process (CC = 0.759, MAE = 0.085).
Figure 4.3: Performance of the computation of a single message in an isolated factor. (a) Schematic of the simulated factor together with the corresponding messages \( \{m_{I1}(t), m_{I2}(t), m_{I3}(t), m_{O}(t)\} \). The factor defines an equality constraint node as defined in [90]. (b) First row: Example spike trains of the three input populations coding for the messages \( m_{I1}, m_{I2}, m_{I3} \) respectively. 343 neurons are coding each message. The green lines schematically indicate the instantaneous firing rates used to create these input spike trains. Every 50ms the rates have been chosen independently for each message according to an uniform distribution. The max./min rates were 90/0Hz, representing the probabilities \( m_{Ik} = 1/m_{Ik} = 0, k \in \{1, 2, 3\} \) respectively. Second row: Response spike trains of a trained readout population representing the output message \( m_{O}(t) \). Third row: blue line: target curve showing the population rate corresponding to the ideal message \( m_{O}(t) \) solving equation 4.3. Red line: actual population rate calculated out of the spike trains in the second row. The encoded probability signal is a linearly scaled version of this rate curve. The correlation coefficient between the red and the blue curve is 0.91.
4.2.3 Inference in Whole Graphs

The ability to train robust individual message outputs, as verified in the previous section, provides a foundation for constructing more elaborate factor graphs with interconnected liquid and readout pools. In this section we therefore apply the scheme of fig.4.1 to two such graphs, both representing inference problems from completely different domains. We first look at a graph stemming from the field of coding theory and which models signal transmission through an unreliable channel. We then construct a FFG modeling a psychophysical 'explaining away' phenomenon [71, 70] and analyze the behavior of LSM networks analogous to both graphs. In addition to the noisy readout population approach, the p-Delta learning rule is also applied for the coding problem, but its use is subsequently abandoned because of the superior performance of the former.

4.2.3.1 A Coding Problem: Inferring Signals From Unreliable Channels

In case of the coding problem it is the principle rather than the particular (e.g. neurophysiological) application that is at issue, and so the corresponding FFG we chose to simulate has been described in the literature as an example of standard discrete BP [90] (Figure 4.4a). In general, such FFGs cannot be converted straightforwardly to the Bayesian Network representation that is typically used in the domain of neuroscience. Hence the coding problem also illustrates the flexibility of our LSM scheme for Belief-Propagation. This problem, expressed as FFG in fig.4.4a, is as follows: Suppose each bit $X_i$ of a binary code word $(X_1, \ldots, X_4)$ is fed into an unreliable signal transmission channel, which inverts the bits with a certain probability. Given the observation of all bits at the end of the channel $(Y_1, \ldots, Y_4)$, we wish to know the marginals $P(X_i | Y_1, \ldots, Y_4)$.

The FFG represents a factorization of the a posteriori probability $P(X_1, X_2, X_3, X_4, Z | Y_1, Y_2, Y_3, Y_4)$, where $Z$ is an auxiliary variable, $X_i$ the value of bit $i$ at the beginning, and $Y_i$ the observed value of this bit at the end of the channel. The product of the '$\oplus$'- and '='-factors connected via $Z$ models membership of a given binary sequence $(X_1, \ldots, X_4)$ to the set of code words above: It can be seen from those code words that the '$\oplus$'-node implements the parity check function $f_\oplus(X_1, X_2, Z) = \delta(X_1 \oplus X_2 \oplus Z)$ ('$\oplus$' denotes the exclusive-or function), and the equality constraint node enforces equality among the variables $Z, X_3, X_4$. The product of the two factors therefore assures that for valid code words both $X_3$ and $X_4$ are the parity bit of $X_1$ and $X_2$. The empty nodes model the unreliability of the bit-wise signal transmission, i.e. they are defined by the conditional probabilities $P_i(Y_i | X_i)$. These probabilities are equal to 0.8 for $X_i = Y_i$ and 0.2 otherwise. The product of all factors yields the a posteriori probability $P(X_1, \ldots, X_4 | Y_1, \ldots, Y_4) \propto f_\oplus(X_1, X_2, Z)f_\omega(X_3, X_4, Z) \prod_{i=1}^{4} P_i(Y_i | X_i)$. By applying
4.2. ENCODING BELIEF-PROPAGATION MESSAGES WITH POPULATION RATE CODES

Message-passing within the Belief-Propagation framework it is possible to obtain the marginal a posteriori probabilities \( P(X_i | Y_1, ..., Y_4) \) for all \( i \) simultaneously, i.e. the degree of belief one can have in the value of each bit at the beginning of the channel, given the observed values of those bits at the end of the channel. The messages are scalar values in \([0, 1]\) representing the sender nodes local estimate about the associated variable to take on value 1. An example message obtained by inserting the definition of the '⊕'-factor into equation 4.1 has already been given in the previous section (eq.4.2).

A network of spiking neurons equivalent to the graph in figure 4.4a is shown in figure 4.4b. Here the readouts were trained by the p-Delta rule. For simplicity only the '⊕'- and the '='-node were explicitly represented by liquid pools. In the first experiment (fig.4.4c) the messages sent by the nodes \( P_1, ..., P_4 \) to the '⊕'- and '='-nodes respectively are constant over time. This corresponds to the case of constant channel transmission probabilities and constant observations of bits \( Y_1, ..., Y_4 \). Roughly after 0.2 seconds the message dynamics have reached steady state, which would allow for a correct extraction of the beliefs \( P(X_i | Y_1, ..., Y_4) \) (not shown). A close match between the desired target and actual message values can be observed. The same finding hold true when the circuit is stimulated with spike trains of temporally varying instantaneous firing rates. The results are shown in fig.4.4d.

However, as already discussed previously, using noisy readout populations yields superior performance compared to the p-Delta rule. From now on, we hence focus exclusively on LSMs endowed with this type of readouts. An appropriate LSM network for the coding problem is illustrated in fig.4.5 (in this case we even simulate liquid pools for the factors \( P_1, ..., P_4 \)). The ability of this neuronal circuit to perform inference is evaluated against a series (n=600) of random external stimulations (corresponding to the messages along the variables \( Y_1, ..., Y_4 \)). For each trial all these external inputs are assigned to a random but temporally constant value (constant instantaneous firing rates). Then for internally computed messages, the error between their exact steady-state value and the corresponding value found by the spiking network is determined (figure 4.5c). Each colored curve shows the histogram of the error \( m_{\text{correct}} - m_{\text{spiking}} \) between the correct steady-state message \( m_{\text{correct}} \) and the one found by the spiking network \( m_{\text{spiking}} \) during each run. The spiking messages are normalized to 1 such that they correspond to probabilities. The black histograms serve as control and correspond to the error \( m_{\text{correct}} - m_{\text{random}} \), i.e. in each run \( m_{\text{spiking}} \) has been replaced by a random probability \( m_{\text{random}} \). The numbers on top of each curve are the trial-averaged Kullback-Leibler (KL) divergences (in bits) between the distributions induced by \( m_{\text{correct}} \) and \( m_{\text{spiking}}/m_{\text{random}} \) respectively. Each run lasts for 0.5s in simulated time. For each message, steady-state is assumed to be reached after 0.2s, the temporal averages of the instantaneous firing rates during the remaining 0.3s are taken as the encoded steady-state solutions found by the spiking network.

The results summarized in Figure 4.5 indicate a close correspondence between the BP steady-state
solution and the solution found by the spiking network. However, the performance decreases with increasing 'depth' of the messages: i.e. the KL divergence and histogram width associated with messages that depend on strongly processed input are typically larger than those of messages that depend on less processed input (compare green, blue/yellow, purple curves with pink/red curves respectively). This effect could be caused by error propagation associated with the necessarily imperfect training of the readouts.

To gain further insight into the dependency between the dynamics as defined by equation 4.1 and those of the neural circuit, we also stimulate the circuit with spike trains of temporally varying instantaneous firing rates. These rates follow, but are different instances of, the same stochastic processes as those used during the training procedure (see section 4.2.1). Since the message dynamics can be important for the BP-algorithm to converge (see section 2.2), the goal here is to assess the goodness-of-fit between a numerical solution of equation 4.1 and the spike dynamics of the readout units when random time-series are externally applied. The correlation coefficients between these two quantities for each message are summarized in figure 4.5d. One can infer a strong linear relationship. However as expected from the previous results, there is a slightly higher correlation for less processed input than for messages 'deep' in the network. Also, coefficients associated with messages emitted by the '⊕'-node are much smaller than those of messages emitted by the '='-node. Probably this result arises because the sum-product rule of the former node is much harder to learn.
4.2. ENCODING BELIEF-PROPAGATION MESSAGES WITH POPULATION RATE CODES

Figure 4.4: 
a) An unreliable binary channel model \[90\] (see text for details). Messages are represented by arrows.
b) Schematic of the neural implementation of the FFG in (a), using interacting pools of neurons (circles). Spiking readouts \(R\) are created by translating p-Delta trained perceptrons \(P\) to IAF neurons. Only the \(⊕\)- and the \(\Rightarrow\)-node are explicitly represented by liquid-pools, the messages emitted by nodes \(P_1, ..., P_4\) are provided by externally applied spike input \(SI\).
c) Perceptron outputs (blue traces) constituting the individual output messages of the \(⊕\)- and the \(\Rightarrow\)-node. Color code of the message names as defined in (a). The red traces give the desired target output for each readout, when the system of coupled differential equations defined by eq.4.1 is simulated numerically. In this case, the input messages emitted by nodes \(P_1, ..., P_4\) are constant over time, which corresponds to the case of constant channel transmission probabilities and constant observations \(Y_1, ..., Y_4\).
d) The same as in (c) but for input messages following Wiener processes. This corresponds to the case of either non-constant transmission probabilities or non-constant observations.
4.2.3.2 Modeling a Biologically Inspired 'Explaining Away' Phenomenon

We now apply the LSM-based BP method to a biologically more realistic inference problem. Many such problems faced by biological systems can be described in an elegant way by using graphical models. For example, [71, 70] define four basic types of Bayesian computations in the framework of object recognition and perception. One of these models the phenomenon of 'explaining away' alternative hypotheses about the interpretation of a visual scene in presence of auxiliary stimuli favoring a particular hypothesis. An example is shown in Figure 4.6a. Knill and Kersten [74] have shown that when two cylinders with an identical photometrical luminance profile are attached to each other, subjects consistently perceive two
identical objects. However, when the cylinders are replaced by cubes with the same luminance profile, the right cube as a whole appears to be brighter than the left one.

The authors explain these distinct perceptions as being due to the integration of auxiliary information provided by the contour shape of the two types of objects. In case of the cylinders, the round contours offer the best explanation for the observed luminance profile, i.e. that the scene has a light source on the left and the two objects have constant material reflectance across position, i.e. across the two cylinders. Thus, due to the assumed same material reflectance, the two cylinders are interpreted and perceived as identical objects. This hypothesis ‘explains away’ an alternative one that assumes uniform illumination, and a higher reflectance of the right object. In the case of straight contours, the latter explanation is more likely and hence the two cubes are perceived differently.

Figures 4.6b and c show a Bayesian Network and its equivalent FFG that provide a probabilistic interpretation of this phenomenon. Figure 4.6d shows our corresponding spiking neural network. The two ‘non-prior’-factors of the FFG in (c) \( P_1(S|R,O) \) and \( P_2(C|O) \) are defined by the values given in tables 4.1 and 4.2 respectively. As for the coding problem above, our goal is to implement general filters as expressed by equation 4.1 for arbitrary input signals \( m_{k→i}(t) \). An example message obtained by inserting the definition of the \( P_1(S|R,O) \)-factor into equation 4.1 can be found in the appendix C.3.

We use the network of figure 4.6d to perform two types of simulations: Firstly, a psychophysically relevant case, in which the visual scene in Figure 4.6a changes suddenly, so that curved contours are observed instead of straight ones (i.e. variable \( C \) in figure 4.6 steps from 0 to 1). We expect this step to result in a decrease in the marginal probability of variable \( R \) (perceived reflectance), indicating the reduced belief in the homogeneous reflectance hypothesis.

Figure 4.7 shows the results obtained by our spiking network, and by direct numerical emulation of BP in that situation. There is a clear decline of the message along \( R \) (green trace) from about 0.6 to about 0.4. Since the prior of \( R \) was assumed to be uniform, this message corresponds directly to the (conditional) marginal probability of \( R \). Therefore, after the transition a constant reflectance across position \( (R = 0) \) is a more likely hypothesis than a reflectance step \( (R = 1) \). On a perceptual level this result corresponds to a change in the interpretation of the scene, i.e. the perception of two different objects switches to the perception of two identical objects. Note the network computes the new steady-state probabilities in approximately 100ms (interval between the grey vertical bars; the LSMs are trained with \( \tau = 10ms \) in equation 4.1). This latency corresponds well to estimates of human performance in object classification tasks (which can also be seen as Bayesian inference problems) as reported in the psychophysics literature [73, 88, 168]. With a classic single-neuron firing rate code such a fast performance would have been rather unlikely to occur, because in this case each computational stage (node) needs to perform temporal averaging of spikes which slows down the dynamics more and more the deeper the node is in the graph.
As in the noisy channel case, we also evaluate the ability of the neuronal circuit to perform inference against a series (n=600) of random external stimulations. The results are summarized in Figure 4.8a. Once again the results indicate a close correspondence between the two solutions which is far from chance level. As before, we evaluate also the correlation coefficients between the spiking and the numerical solutions after applying random input time-series, which give the results of figure 4.8b. There, we can also see a slight effect of error accumulation which gives rise to smaller correlation coefficients for strongly processed messages (e.g. compare $m_{P_1 \rightarrow P_3}$ to $m_{P_1 \rightarrow s}$). It can also be seen that for this graph the minimum correlation coefficient of all messages is larger than in the noisy channel example. Most likely this is because the messages emitted by factor $P_1$ in the 'explaining away' graph are easier to learn than those emitted by the '$\oplus$'-node in the noisy channel case.

<table>
<thead>
<tr>
<th></th>
<th>$O = 0$</th>
<th>$O = 1$</th>
<th></th>
<th>$O = 0$</th>
<th>$O = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R = 0$</td>
<td>0.8</td>
<td>0.2</td>
<td>$R = 0$</td>
<td>0.2</td>
<td>0.8</td>
</tr>
<tr>
<td>$R = 1$</td>
<td>0.2</td>
<td>0.9</td>
<td>$R = 1$</td>
<td>0.8</td>
<td>0.1</td>
</tr>
<tr>
<td>$S = 0$</td>
<td></td>
<td></td>
<td>$S = 1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: Factor entries for $P_1(S|R,O)$

<table>
<thead>
<tr>
<th></th>
<th>$O = 0$</th>
<th>$O = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C = 0$</td>
<td>0.8</td>
<td>0.2</td>
</tr>
<tr>
<td>$C = 1$</td>
<td>0.2</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 4.2: Factor entries for $P_2(C|O)$
4.2. ENCODING BELIEF-PROPAGATION MESSAGES WITH POPULATION RATE CODES

Figure 4.6: Graphical models and neural implementation of an ‘Explaining Away’ phenomenon. (a) Knill and Kerstens lightness illusion (see text for details, figure taken from [2], see http://web.mit.edu/persci/gaz/gaz-teaching/index.html for an animated movie making the illusion even more apparent). (b) Bayesian Network describing the illusion in (a) as an explaining away phenomenon. All variables (nodes) are binary. The observed luminance profile or shading can either be as in (a) ($S = 1$) or different ($S = 0$). The material reflectance of the two objects can either be homogeneous ($R = 0$), or such that the reflectance of the right object is higher ($R = 1$). There are two types of 3D objects, cylinders ($O = 1$) and cubes ($O = 0$), and two types of observed contours: curved ($C = 1$) and straight ($C = 0$). The network corresponds to the joint probability distribution $P(S, R, O, C) = P_1(S|R, O)P_2(C|O)P_3(R)P_4(O)$. In this model, whether two attached objects are perceived identically or distinctly will depend on the marginal probability of the material reflectance $P(R|S, C)$ given external observations $S$ and $C$ (graph taken slightly modified from [70]). (c) FFG representing the same joint probability as in (b). Observed variables $S$ and $C$ occur as singly connected edges. (d) Implementation of the Belief Propagation algorithm with spiking neurons on the FFG shown in (c). The individual liquid pools ($L_i$) correspond to the factors labeled by the same subscript $i$. Together with the associated spiking readouts ($R_i$) message signals are implemented. Externally provided spike inputs ($SI_i$) either stand for observed variables ($C$ and $S$) or for constant messages originating from factors representing prior knowledge ($P_3(R)$ and $P_4(O')$). The purpose of the network is to compute all the (conditional) marginal probabilities associated with the unobserved variables.
Figure 4.7: Simulating ‘Explaining Away’ with a sudden change in contour perception. (a) Color code of the messages (arrows) within the network of Figure 4.6. Externally supplied input messages are fed into the network via the black SI-populations. All inputs were constant during the whole run besides the message entering node $P_2(C|O′′)$ along observed variable $C$. This probability switched from 0.1 to 0.9 at time 0.5s, indicating the visual scene has suddenly changed from observed straight to curved contours. External input messages from nodes representing prior knowledge were uniform ($m_{P_3(R)→P_1(S|R,O)}(t) = m_{P_4(O′)→P_2}(t) = 0.5$) and a nonuniform luminance profile was observed ($m_{S→P_1(S|R,O)}(t) = 0.9$). (b) Colored solid lines: Time-series of the probability values decoded from the population rate of the individual readouts in response to the externally applied spike trains representing the input messages. Color code as in (a). Black dashed lines represent the result of a non-neural simulation of BP by solving equation 4.1 numerically.
Figure 4.8: (a) Performance analysis of the ‘Explaining Away’ circuit. Shown are the results of a Monte-Carlo simulation consisting of 600 runs. In each run, all external spike inputs (‘SI’ in figure 4.7) were assigned to randomly chosen, constant values (constant instantaneous firing rates) corresponding to random input messages. Each colored curve shows the histogram of the error $m_{\text{correct}} - m_{\text{spiking}}$ between the correct steady-state message ($m_{\text{correct}}$) and the one found by the spiking network ($m_{\text{spiking}}$) during each run. Since the messages are normalized to 1, they correspond to probabilities. 100 bins were used, color code of the messages is as in figure 4.7. Each run lasted for 0.5s in simulated biological time. Steady-state was assumed to be reached after 0.2s, the temporal averages of the instantaneous firing rates during the remaining 0.3s were taken as the encoded steady-state solutions found by the spiking network. The black histograms serve as control and correspond to the error $m_{\text{correct}} - m_{\text{random}}$, i.e. in each run $m_{\text{spiking}}$ has been replaced by a random probability $m_{\text{random}}$. The numbers attached to each curve are the trial-averaged Kullback-Leibler divergences (in bits) between the distributions induced by $m_{\text{correct}}$ and $m_{\text{spiking}}/m_{\text{random}}$ respectively. The trial-averaged entropies of the correct distributions were between 0.48 and 0.67 bits. (b) Correlation coefficients for each message in (a) between a numerical evaluation of continuous time BP and the population rates of the readout pools. External input was provided by instances of the stochastic process as defined in section 4.2.1. These time-series had a length of 20s in simulated time. The correlation coefficients were highly significant with a p-value less than 1%.

<table>
<thead>
<tr>
<th>Message</th>
<th>$m_{p1 \rightarrow p3}$</th>
<th>$m_{p1 \rightarrow s}$</th>
<th>$m_{s \rightarrow p1}$</th>
<th>$m_{s \rightarrow p2}$</th>
<th>$m_{p3 \rightarrow p4}$</th>
<th>$m_{p2 \rightarrow s}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>0.73</td>
<td>0.85</td>
<td>0.94</td>
<td>0.97</td>
<td>0.94</td>
<td>0.99</td>
</tr>
</tbody>
</table>
4.3 Chapter Summary & Discussion

Based on Liquid-State Machines we have presented a general strategy for implementing message-passing algorithms in spiking neural circuits (see figure 4.1). The principle is to use reservoirs composed of pools of spiking neurons as nodes in graphs. Each pool gathers messages from its neighbors and combines those through its network dynamics. The various output messages transmitted over the edges of the graph are extracted from their source liquids by groups of readout neurons and fed into their destination liquid pools. This approach is very flexible because, by using a separate set of readouts, it not only provides a neural substrate for various different message-passing algorithms, but also allows for the computation of other quantities of interest that depend on the exchanged messages, but which themselves do not influence the computation of these messages. Furthermore, in principle it also allows for the embedding of multiple graph structures and node parameterizations using the same neural machinery. Within the scope of this thesis, we have shown as a proof of concept how this approach can be used to implement the Belief-Propagation algorithm in networks of spiking neurons, that process BP messages on the basis of population rates. We have emulated an inference problem from the domain of coding theory, dealing with the passing of bits through an unreliable signal transmission channel (figure 4.5), and also a more biologically relevant case, in which visual cues are used to resolve hypotheses about an object’s shape and illumination (figure 4.6). The dynamics of the respective networks followed closely a numerical evaluation of the BP algorithm.

The mechanism that we have presented here is very general, and lends itself to implementation in, for example, the neuronal networks of the neocortex. Many anatomical studies indicate that the entire neocortex is composed of a relatively few basic types of excitatory and inhibitory neurons that are everywhere connected in a similar pattern, suggesting that cortex uses a common fundamental circuit architecture that repeats over its entire area. Various candidates for the computational process supported by this common architecture have been proposed (e.g. 62, 133, 101). But, the ability to infer unknown information from incomplete sensory data combined with some prior knowledge must rank as one of the most fundamental principles for incorporation in cortical circuits. And here we have shown that such inference can be instantiated in a highly distributed manner, using a common architecture of interconnected pools of spiking neurons (figure 4.1).

As stated above, in our simulations we have used ‘general purpose’ liquid pools which, when interpreted biologically, opens the possibility for cortex to perform many other computations on top of and in parallel to BP using the same neural hardware. Therefore, our approach predicts that, on the anatomical level, there is no specific neural circuitry implementing the Sum-Product rule. Instead, it can just be one out of many computational results that a specific readout can extract from a generic pool. For example, the
most probable state of each marginalized variable connected to a given liquid-pool might be determined by BP, and the resulting state-vector then be processed by a classifier readout unrelated to the inference task at hand. Such readouts could be represented by projection neurons transmitting their classification results to various target areas.

On the other hand, if cortex is a pure inference machine, a ‘general purpose’ liquid pool might be unnecessarily complex for only inference tasks. An interesting future direction of research is therefore the development of an appropriately ‘structured liquid pool’, whose computational abilities are specialized to the supply of all necessary products according to the Sum-Product rule. In a different information processing context Häusler and Maass [57] have already investigated this idea using liquid-pools with a lamina-specific connectivity structure that respects experimentally assessed connection statistics. For a variety of computational tasks they have found this circuit to yield superior results compared to random liquid-pools. In case of the Sum-Product rule such a ‘special purpose’ circuit might be genetically prewired in cortex, and can be thought as an explicit multiplicative feature mapping whose scalar product with itself implements a ‘kernel function’ as in the framework of support-vector-machines [15]. The Sum-Product rule would then be provided by the output of a trained readout through a linear combination of the liquid-provided products. In this context a factor node is exclusively defined by the weights of its readouts, and a customization of the factor to a particular problem can then be accounted for by assigning specific values to those weights.

This approach raises the question whether it is possible to learn these weights by un- or semi-supervised (reinforcement) learning methods. Future work might include the use of reward-modulated STDP learning to obtain suitable readout weights [85][65]. This method has already been applied successfully to responses of a generic cortical microcircuit (liquid) model [85].

In the context of BP one might ask the following natural question with respect to the neural architecture in figure 4.1: Given the approximation properties of LSMs [101], would it not be possible to perform the whole message-passing in a single liquid pool, with each of its readouts feeding back into the pool in a recurrent manner, rather than using a distributed arrangement of LSMs? These readouts could also be trained according to the Sum-Product rule (equation 2.9). Although this strategy looks plausible, there is a clear reason why the distributed arrangement is superior: In BP every node uses exclusively locally available incoming messages to compute its outgoing messages. In the LSM framework this means that each readout is required to access only a subset of all shared knowledge within the network. Using a single global liquid pool with recurrent readouts makes the state of the liquid neurons dependent on all information present in the network. Therefore, as the number of nodes in the graphical model increases it becomes more difficult to filter out irrelevant information during training of the readouts.

In using pools of neurons, our approach stands in contrast to previous models whose basic inference
elements are single neurons [13, 12, 32]. The advantage of interacting pools is that they allow for an implementation of FFGs, which subsume Bayesian Networks (as in [13, 32]) as well as Markov Random Fields (as in [12]). Bayesian Networks and Markov Random Fields in contrast are restricted in that there are instances of graphs of either type that cannot be translated into the corresponding other one, without the introduction of additional edges [76]. These edges may not only lead to higher dimensions of the factor functions, which are more difficult to represent neurally, moreover, they may render a tree-shaped graph loopy.

However, a restriction of the case studies presented in this chapter is their exclusive use of binary variables. If normalized messages are assumed, binary variables allow each message (i.e. probability estimate) to be defined by a single scalar value, and so only a single readout population is needed to encode this value. In principle though, this population-based readout method is extendable to multivalued variables by using a separate population for each individual value of such a variable. Unfortunately, we expect this strategy to be expensive in terms of the number of required neurons, both for the liquid and readout populations. A problem of a rate-based population code is that only the fraction of neurons firing during a small time interval, and not their position within the population, is the relevant variable. This restriction hinders the learning of the readouts, and we assume that it is part of the reason for the larger sizes of our liquid pools compared to those reported by Maass [101] (see appendix section C.1). Therefore a place coding scheme, which is also closer to known properties (tuning curves) of real neurons, could be a further extension of the presented approach.
Chapter 5

Results III: Casting Abstract Models For Spike-Based Belief-Propagation Into Networks of Neural Elements

So far we have considered separately the abstract and concrete spiking Belief-Propagation (BP) models of chapters 3 and 4 respectively. While the former allowed for very general (e.g. analog) versions of BP, it was not clear how these models could possibly be cast into a simplified neural machinery. Conversely, the concrete spiking BP model was purposely designed to match the computational prerequisites of (spiking) Liquid-State Machines (LSMs). In practice however its representational power was restricted to Forney-Factor Graphs (FFGs) with binary variables. Therefore, the goal of this chapter is to blend together the advantages of both kinds of models, by embedding the abstract BP processors into our general reservoir computing framework for message-passing (section 4.1). We first show how this can be achieved for the abstract ISI processor of section 3.2. Implementation results of the approach of 3.3 are presented subsequently. It is important to note however that these latter attempts have been much less successful than those of the ISI approach.

5.1 Concrete Neural Architecture for the Abstract Belief-Propagation Processor of Section 3.2

Although the abstract, functional description of the ISI processor presented in section 3.2 is spike-based, it is not obvious, (a) what principal computational architecture/learning machine using what learning
algorithm can be used to approximate that processor, and (b) what biophysical, neural mechanisms in turn could possibly be exploited to concretely implement that machinery in neural systems. Based on our reservoir computing framework for general message-passing (see section 4.1), we here provide an answer to question (a). However, regarding question (b), the resulting approach is flexible enough to be consistent not only with its straightforward interpretation as a network of neurons, but also with processing inside the dendritic tree of a single neuron. This stands in marked contrast to the population rate approach of section 4.2 and will be discussed in detail in the chapter summary (section 5.3).

After explaining the computational architecture in detail, we show performance evaluations for the case of computing single output messages. Consequently it is then verified that this output can be used as input to a subsequent processor stage -a situation that necessarily occurs when inference is conducted in full graphs.

5.1.1 Functional Description of the Architecture

Slightly different from the LSM network architecture of section 4.2, the model of figure 5.1 provides an answer to question (a). Again, without loss of generality, we illustrate the approach for the case of computing a single output message in a 3-way factor node. The architecture of fig.5.1 thus serves as generic neural model for computing single ISI-encoded output messages of a node \( m_{out} \) from the corresponding input messages \( m_X, m_Y \). It is conceptually divided into two parts. In the first part the incoming spike trains representing \( m_X \) and \( m_Y \) are separately decoded to time-series of analog ISI values. That is, each time there is an input spike along any message input, the corresponding decoder jumps to the new ISI value that labeled the spike and holds this value until the next spike comes in. Thus, both decoders as a whole constitute a preprocessing block that serves the purpose of positional sampling (see the abstract ISI processor description of figure 3.3). The only difference is that here positional sampling necessarily happens instantaneously, i.e. only the last sample \( (x,y) \) on the \( (X,Y) \)-plane is represented and there is no temporal sliding window \( W \) involved.

Using the decoded ISI time-series, the functionality of this window is delegated to the second part of the network architecture. In addition this part also performs the remaining computations needed by the abstract ISI processor (figure 3.3). Following the reservoir computing paradigm, both parts are implemented by Echo-State Networks (ESNs), i.e. the basic units in the reservoirs (illustrated as grey 'L' circles) are (leaky) sigmoidal not spiking neurons (see for details of the neuron model). We have utilized this framework rather than LSMs, because for many tasks ESNs are easier to work with, i.e. to set up and tune the parameters. Furthermore, as mentioned above, the abstract ISI processor provides a biophysical interpretation other than a network of spiking neurons. As discussed later in this thesis the dendritic tree of a single neuron might also act as a reservoir, in which case sigmoidal units are
5.1. ANALOG BELIEF-PROPAGATION USING INTERSPIKE INTERVALS

Depending on the task, the individual readouts in fig.3.3 are of different type: The ISI decoding readouts $R_X$ and $R_Y$ are linear, simply performing a weighted sum of their inputs from $L_X$ and $L_Y$ respectively. Correspondingly, their input weights are determined by ordinary linear regression. These readouts can also chosen to be sigmoidal units, provided the dynamic range of represented ISIs falls within the linear part of the sigmoidal curve. In addition to providing input to the SPR computation stage (reservoir $L_{out}$), the output of $R_X$ and $R_Y$ also feeds back to their respective reservoir. As mentioned in section 2.4 this drastically prolongs the memory capacity of the ESNs. Ideally, if perfect, errorless performance on test sets could be achieved after training, such feedback would allow even for nonfading memory computations [99]. Long memory is indeed needed by the ISI decoding readouts, since their task is to detect the ISI value associated with each spike and to hold that value until the next spike arrives.

In contrast to $R_X$ and $R_Y$, readout $R_{out}$ constitutes the final output of the processor and hence belongs to some leaky-integrate-and-fire (LIF) neuron model with escape noise [55]. The membrane voltage $V_m$ of $R_{out}$ behaves linearly, i.e. it is given by the weighted sum of activations of the sigmoidal neurons in $L_{out}$. Subject to escape noise the neuron may fire spikes at any time $t$, with a firing hazard $h(t)$ that depends exponentially on $V_m(t)$: $h(t) = h(V_m(t)) = \frac{V_m(t) - V_\theta}{\sigma_h}$ (with $h_\theta$ denoting the hazard at the (soft) threshold value $V_\theta$ and $\sigma_h$ determining the 'softness' of the threshold). This type of hazard activation function for spike firing is often used in the neuroscientific literature [128, 127, 136, 37]. A thus fired spike is fed back into the recurrent network $L_{out}$. According to the processor description of fig.3.3 this feedback is necessary to signal the circuit when the function spike labeler and probability integrator have to be reset. In contrast to $R_X$ and $R_Y$, readout $R_{out}$ is trained by minimizing the summed Kullback-Leibler (KL) divergences between target and actual ISI distributions using the BFGS quasi-Newton method [12]. We will explain later in the chapter summary why, unlike $R_X$ and $R_Y$, linear regression cannot be used here. However, instead of batch-mode quasi-Newton methods, it is also possible to employ a STDP-like learning scheme [128], which amounts to a stochastic gradient descent procedure [19] applied to the negative log-likelihood of observed spike samples. Interestingly, under rather loose and biologically plausible conditions imposed on $h(V_{in})$, this second cost-function is convex. See appendix section D.1 for a detailed account of both functions and a proof of convexity for the latter.

Spike-based inputs to the sigmoidal neurons of $L_X$, $L_Y$ and $L_{out}$ are provided by low-pass filtering of the spikes, i.e. by convolving the spikes with an exponential kernel of height 1. This makes discrete time-series of point events compatible with analog processing units and moreover mimics postsynaptic potentials.

Empirically we have found the approach not to work when the ISI decoder readouts are removed and the low-pass filtered spikes from $m_X$ and $m_Y$ are fed directly into $L_{out}$. Presumably this is due
to the abundance of spike-timing invariances the network has to learn in this case. Indeed, there are many instances of time-shifted spike trains that all lead to the same sequence of (integral) function spikes within temporal sliding window \( W \). Following the sampling scheme of figure 3.2, as long as the ordering of spikes is not affected, i.e. the sequence of channel identities \( \{X,Y\} \) that indicate for each (integral) function spike the channel that caused it, any shift between the spike trains \( m_X \) and \( m_Y \) yields the same sequence of (integral) function spikes. Hence, also the ISI distribution \( p(\Delta t, t) \) that is to be learned remains invariant in this case (eq. 3.5). Invariance to shifts of spike trains however means being invariant to relative spike timings and this is precisely what the network is not allowed to be due to a) the need for ISI decoding and b) the dependence of the function spike labeler and probability integrator on spikes from \( R_{out} \). Therefore, there is an inherent design conflict when the reservoir parameters have to be tuned: As far as inputs \( m_X \) and \( m_Y \) are considered in combination, the reservoir parameters have to be adjusted in such a way that learning leads to (relative) spike timing invariance. On the other hand such invariance is forbidden for spikes from \( R_{out} \) and for the individual decoding of ISIs from spikes of \( m_X \) and \( m_Y \). Having dedicated ISI decoder units resolves the conflict, because the time-series they produce represent directly the sequence of ISIs as an analog signal, for which timing is not an issue. Hence reservoir \( L_{out} \) can safely be tuned to become sensitive to spike timings (for spikes from \( R_{out} \)).

We now turn to simulating the architecture of figure 5.1.
5.1. ANALOG BELIEF-PROPAGATION USING INTERSPIKE INTERVALS

5.1.2 Performance Analysis

In the spirit of section 3.2.2 we now conduct a series of simulations evaluating the performance of an ESN based processor as shown in figure 5.1. For that the ESN software toolbox of Jäger is used (publicly available at [http://www.reservoir-computing.org/](http://www.reservoir-computing.org/)) and customized where needed. Settings of the various reservoir parameters are listed in the appendix (section D.2).

The processor is trained to approximate a BP message emitted by a factor node $f_1$ that consists of an unnormalized gaussian mixture (four components with equal weight 0.5; random mean vectors whose components are drawn independently and uniformly from $[15, 60]$; circular covariance matrices, $\sigma = 7$). After training, the ESN network is stimulated by a series of spike input messages which are represented by various different ISI distributions. These distributions are also created by random gaussian mixtures, having 1 to 3 mixture components (figure 5.2). Since only a single message is simulated, the target belief...
of the output variable according to equation 2.12 cannot be computed. Therefore, the normalized output message $m_{out}$ based on equation 2.11 plays the corresponding role. As in section 3.2.2 the approximation quality is apparent.

We now verify whether two serially connected ESN processor stages can compute the correct BP output message. This test is mandatory to see if the architecture of figure 5.1 allows for a modular composition of larger FFGs, by enabling the desired communication between two adjacent factor nodes. Therefore, the output of $f_1$ shown in figure 5.2 is used as input to a subsequent factor $f_2$ along one of $f_2$’s variables. To verify proper intercommunication even between factor nodes belonging to different families of functions, factor $f_2$ is parameterized in a slightly more general way than $f_1$: The covariance matrices of its four Gaussians are also randomized through uniformly drawn principal component directions, whose associated standard deviations are independently and uniformly taken from $[3, 10]$.

The result of such serial assembly of factors can be seen in figure 5.3. The performance is slightly worse compared to the single factor evaluations of $f_1$ (worst case KL-divergences are approximately 0.21 and 0.15 bits respectively), which is mainly due to two effects: Firstly, the processor based on $f_2$ is more difficult to learn because of the more complicated covariance structure of its gaussian mixture components. Secondly, there is the effect of error propagation when the output of $f_1$ is used as input for $f_2$. Furthermore, the expected KL-divergences used as a control are lower on average in case of serial connection. This finding is independent of the processor approximation quality and is caused by the smoothing of messages when the SPR is iteratively applied to factors constructed as random gaussian mixtures. The same effect can be observed for the belief of variables deep in the graphs of figures 3.5 and 3.6. However considering the logarithmic scaling of the y-axis, the overall performance degradation in the serially connected case is minor.
5.1. ANALOG BELIEF-PROPAGATION USING INTERSPIKE INTERVALS

Figure 5.2: Performance results of a single ESN based BP processor as shown in figure 5.1
Top row: Approximation quality of the BP output message $m_{out}$ when input messages $m_X$ and $m_Y$ were both created by random gaussian mixtures consisting of (a) 1, (b) 2 and (c) 3 mixture components. x-axis: Trial number, y-axis: Values of $D(m_{out}||m_{out,target})$ (black dots, in bits) and $D_{norm}(m_{out}||m_{out,target})$ (red dots). Blue bars indicate the expected value and 3 times the standard deviation of $D(m_{out}||m_{out,target})$ when $m_{out}$ is replaced by a random distribution. In (a) and (b) all mixture components had equal weight, in (c) weight proportions were drawn randomly from $[0,2,1]$. The components’ widths were a) $\sigma = 1.5$, b) $\sigma = 2.5$ and c) $\sigma = 1.5$
Bottom row: Example ISI histograms of $m_{out}$ for individual trials in each of the three cases (blue bars). Red curves show $m_{out,target}$.
5.2 Concrete Neural Implementation of the Abstract Belief-Propagation Processor of Section 3.3

In the spirit of the previous section we here combine the abstract event-based BP-processor of section 3.3 with our general reservoir computing framework for message-passing in neural systems (section 4.1).

We begin by presenting results from successful implementations of simple 2- and 3-way factor nodes. Subsequently, we show data illustrating our unsuccessful attempts of emulating a more complicated 3-way factor node. In this case we have also (unsuccessfully) tried to involve methods for tuning the reservoir.
5.2. DISCRETE BELIEF-PROPAGATION USING GENERAL SPIKE-BASED EVENTS

5.2.1 Simulation of a 2-Way Factor Node

We take a simple gaussian \( f(X,Y) \) as it frequently occurs in models of noise in sensory systems [38, 77] as our first example of a 2-way factor node depending on variables \( X \) and \( Y \): 

\[
f(x,y) \propto \exp \left[ -\frac{(x-y)^2}{2\sigma} \right]
\]  

Equation 5.1

\( X \) and \( Y \) are each discretized into six states \( x,y \in \{1,\ldots,6\} \), where \( X \) is considered the input- and \( Y \) the output variable. Only the output message along the output variable is computed. States of both variables are neurally represented by spike trains coming in and out of the LSM respectively. We use the presence of at least a single spike in time windows of length \( W \) as event definition (eq. 3.31 for \( n = 1 \)). Because a BP output message does not depend on the input message coming in along the same variable, there are no input spike trains representing the input message along \( Y \). Each state of input variable \( X \) is represented by three independent spike trains, whereas the states of \( Y \) have ten spike trains each. The reference input consists of a single train. Therefore, in total the reservoir receives \( 6 \times 3 + 1 = 19 \) inputs and provides output through a population of \( 6 \times 10 = 60 \) spiking readouts, where all ten readouts belonging to the same subpopulation share the same set of synaptic weights. Multiple spike trains per variable state are used to obtain a larger number of spike samples per time, which allows for faster assessment of the BP messages.

To determine the target time series used for training of the individual readouts, the following procedure has been conducted. After each spike at time \( t_i \) at the reference input, a pulse kernel \( r^*(t-t_i,t_i) := F_i^*(y) \cdot r_0(t-t_i) \) was added to the instantaneous firing rate profile \( r(t) \) of all the readout neurons representing state \( y \) (see section 3.3). We have used kernels of the form

\[
r_0(s) := \frac{1}{\tau_1 - \tau_2} \left[ \exp \left( -\frac{s}{\tau_1} \right) - \exp \left( -\frac{s}{\tau_2} \right) \right] \cdot \Theta(s)
\]

Equation 5.2

where \( \Theta(s) \) denotes the Heaviside step function. Using \( r(t) \) we then took the inverse of the diffusive noise model 2.25 to compute the input current \( I(t) \) of the readouts. This current poses a suitable target for learning readouts of the integrate-and-fire type, since, together with a (diffusive) white noise input, it induces the desired instantaneous rate profile \( r(t) \) in such neurons. Determination of the inverse of model 2.25 was done by simply interpolating between \((I,r)\) pairs that had been precalculated according to the monotonic relationship 2.25.

Using \( I(t) \) as target, the readout weights (equations 2.20 and 2.28) can subsequently be computed by linear regression, as in case of the population rate model of section 4.2. In terms of instantaneous firing rates, figure 5.4 shows for testing data the performance of a LSM trained in this fashion.
(see appendix section D.3 for used parameters). The first 2s out of a total simulation time of 6s are shown. Inputs along $X$ are given by Poisson spike trains with rates corresponding to the input message \{u_X(1),...,u_X(6)\} = \{0.85, 0.15, 0.15, 0.15, 0.15, 0.85\}. Figure 5.5\(\textbf{b}\) shows the corresponding read-out spike trains and the decoded message values \{m_Y(1),...,m_Y(6)\}. Despite the fact that squashing function 3.31 is used to compute the output message (which distorts the result of the Sum-Product rule), it is apparent from (b) that the network is still able to put out a close-to-optimal result. Most likely this is due to the low average firing rates of the readout neurons (1–4Hz in fig 5.4), which shifts the operating point close to zero and hence to the linear part of the squashing function. As explained in section 3.3 this regime can always be selected by uniformly downscaling of the entries in the factor table.

Figure 5.4: Performance after learning message outputs from a simple gaussian 2-way factor node function. Shown are the instantaneous firing rate profiles of the six readout populations \(R_1,...,R_6\) representing states 1,...,6 of variable \(Y\) (blue curves). The blue traces were computed on testing data, by applying the diffusive noise model 2.25 to the synaptic input current of the readouts. The corresponding target rates are shown in red. The \(CC\) values give the Pearson Correlation Coefficient between these two traces.
5.2. DISCRETE BELIEF-PROPAGATION USING GENERAL SPIKE-BASED EVENTS

5.2.2 Simulation of a 3-Way Factor Node

We now investigate the more complicated scenario of simulating a LSM circuit that emulates an output message of a 3-way factor node attached to variables $X, Y, Z$. The factor function is defined by the sum of three gaussians, each with isotropic covariance matrix, uniform weight and random mean. This time, input messages along each of the variables are fed into the reservoir, such that in principle the same reservoir could be used to compute all output messages. However, we here show only results from computing the output message along $Z$. Again, all variables have the same domain $x, y, z \in \{1, ..., 6\}$, but each such state is now represented by six neurons. There is no separate reference input and the spike trains representing message inputs along $Z$ play the corresponding role, since otherwise they are of no use for computing $Z$’s output message. Using as reference input the messages coming in along the same variable for which the output message is computed poses a more economic strategy, because in this case the activity vector of the reservoir does not depend on inputs the target function of any of the readouts

Figure 5.5: Readout output and BP message approximation for the gaussian 2-way factor node
a) Spike raster plot of readout subpopulations $R_1, ..., R_6$. For each of the ten neurons per subpopulation, all spike times are marked by a dot.

b) Green bars: Ideal normalized output message $m_{\text{out}}(y)$ for each state $y \in \{1, ..., 6\}$ as computed directly from the factor function and the input message $\{um_X(1), ..., um_X(6)\}$. Red bars show the output message computed by applying squashing function 3.31 to the integrated target instantaneous firing rate (= red traces in fig. 5.4) and normalizing. The output message based on the actual (squashed) integrated instantaneous rate (= blue traces in fig. 5.4) is indicated by the blue bars, whereas the magenta bars give the same for rate estimation based on the spike data in (a). $D_{\text{norm}}$ is the percentage of superfluous bits due to optimal code construction based on the distribution represented by the blue bars, when the true distribution is assumed to be given by the green bars (see section 3.2.2). 'Control' indicates the Monte-Carlo average of $D_{\text{norm}}$ when the blue distribution is replaced by a random distribution taken uniformly from the 6-dimensional probability simplex.
is independent of.

As before, each readout subpopulation represents one state of the output variable and the set of weights is identical for all six neurons in the same subpopulation. Hence in total the reservoir receives $3 \times 6 \times 6 = 108$ input spike trains and gives rise to $6 \times 6 = 36$ readout neurons. Figure 5.6 shows again the instantaneous rates during the first 2s of a 6s simulation run, where each component of the input messages $um_X(\cdot)$ and $um_Y(\cdot)$ is taken randomly from the interval $[0.1, 0.9]$. As expected the CC performance is slightly decreased compared to the single gaussian 2-way factor, because the problem of learning a gaussian mixture 3-way factor is significantly harder. Also, when the message encoded by the readouts is examined (fig. 5.7), there is a marked increase in percentage of superfluous bits compared to the simple gaussian 2-way factor (fig. 5.5). However, the results of fig. 5.7 are obtained after squashing function 3.31 has been applied to the averaged rates (which are between 3 and 18Hz), hence so far one cannot be sure if the observed decrease in performance can be attributed mainly to the more difficult learning of the readouts, or to the effect of the squashing function. Indeed if squashing of the rates is spared, approximation performance becomes comparable to the 2-way factor (fig. 5.8). Intuitively a performance this good is a surprising result, given the rather mediocre approximation performance of the instantaneous firing rates (fig. 5.4, 5.6). Most likely this is due to eq 3.30, which indicates that for decoding $uSPR$ fluctuations in instantaneous rates are unimportant long as the mean rates are matched.
Figure 5.6: Performance after learning message outputs from a gaussian mixture 3-way factor node function
Shown are the instantaneous firing rate profiles of the six readout populations $R_1, ..., R_6$ representing states 1, ..., 6 of variable $Z$ (blue curves). The blue traces were computed on testing data, by applying the diffusive noise model 2.25 to the synaptic input current of the readouts. The corresponding target rates are shown in red. The $CC$ values give the Pearson Correlation Coefficient between these two traces.
Figure 5.7: Readout output and BP message approximation for a 3-way factor node

a) Spike raster plot of readout subpopulations \( R_1, \ldots, R_6 \). For each of the six neurons per subpopulation, all spike times are marked by a dot.

b) Green bars: Ideal normalized output message \( m_{out}(z) \) for each state \( z \in \{1, \ldots, 6\} \) as computed directly from the factor function and the input messages \( \{um_X(1), \ldots, um_X(6)\} \) and \( \{um_Y(1), \ldots, um_Y(6)\} \). Red bars show the output message computed by applying squashing function 3.31 to the integrated target instantaneous firing rate ( = red traces in fig. 5.6) and normalizing. The output message based on the actual (squashed) integrated instantaneous rate ( = blue traces in fig. 5.6) is indicated by the blue bars, whereas the magenta bars give the same for rate estimation based on the spike data in (a). \( D_{norm} \) is the percentage of superfluous bits due to optimal code construction based on the distribution represented by the blue bars, when the true distribution is assumed to be given by the green bars (see section 3.2.2). 'Control' indicates the Monte-Carlo average of \( D_{norm} \) when the blue distribution is replaced by a random distribution taken uniformly from the 6-dimensional probability simplex.
5.2. DISCRETE BELIEF-PROPAGATION USING GENERAL SPIKE-BASED EVENTS

Superfluous Bits: $D_{\text{norm}} = 0.43\%$, Control $= 26.89\%$

Figure 5.8: Using unsquashed readout rates to approximate a BP message of a 3-way factor node
Green bars: Ideal normalized output message $m_{\text{out}}(z)$ for each state $z \in \{1, \ldots, 6\}$ as computed directly from the factor function and the input messages \{$um_X(1), \ldots, um_X(6)$\} and \{$um_Y(1), \ldots, um_Y(6)$\}. Red bars indicate the message estimate based on the scaled, unsquashed integrated instantaneous rates of the learning target ( = red traces in fig. 5.6), where the scaling is due to normalization of the message. The message estimate induced by the actual integrated instantaneous rate ( = blue traces in fig. 5.6) is shown by the blue bars, whereas the magenta bars give the same actual spike data. $D_{\text{norm}}$ is the percentage of superfluous bits due to optimal code construction based on the distribution represented by the blue bars, when the true distribution is assumed to be given by the green bars (see section 3.2.2). 'Control' indicates the Monte-Carlo average of $D_{\text{norm}}$ when the blue distribution is replaced by a random distribution taken uniformly from the 6-dimensional probability simplex.

5.2.3 Verifying Input-Output Consistency

A sensitive issue is whether the spike output of the readouts preserves the event-based coding scheme that underlies the input messages, i.e. whether the probability $P(s_{z,t} = 1) = um_{\text{out}}(z)$ of having at least one spike in $W$ still obeys eq. [3.31]. As with all other models in this thesis, input-output consistency must be guaranteed to ensure that, in a whole graph, a factor node can make use of messages from its neighbors. In the present case, event-coding is preserved if the readout spike trains can be reasonably modeled as homogeneous Poisson processes. A priori however this assumption is challenged by at least two arguments: a) As mentioned before, instantaneous rates are fluctuating due to the adding of pulse kernels after each reference spike, which renders the spike process inhomogeneous. b) Even if rates were constant, diffusive noise can still lead to more regular spiking that is far from Poisson, provided the rates are sufficiently high [155, 6]. Qualitatively the latter is due to the large mean input current that is needed to allow neurons to fire at a high rate (i.e. due to the monotonic $r(I)$ relationship [2.25], because a larger...
and larger mean current renders noise more and more negligible, leading to increasingly regular spike trains that approximate the noiseless case.

Because of concerns (a) and (b) we now verify the validity of the Poisson model in the following way: If the readout spike trains indeed follow Poisson processes, the number of spikes $n$ within windows of size $W$ should be a Poisson distributed integer. Hence we can draw a set of random time points $\{t_i \mid i = 1, ..., m\}$, for which the number of spikes in $[t_i - W, t_i]$ is determined, which thus leads to a set of $m$ samples of $n$. Using measure $D_{\text{norm}}(P^* \| P)$ from section 3.2.2 the corresponding sampling distribution $P^*(n)$ can then be compared to a Poisson distribution $P(n \mid r, W) = \frac{(rW)^n}{n!} \cdot \exp[-rW]$, where $r$ is the empirical rate of the readout spike train. $m$ should be large in order to avoid errors that are caused by undersampling. An additional measure for ‘poissonness’ of spike trains is the Fano Factor, which is 1 in the ideal Poisson case and which is often used to evaluate real spike trains from electrophysiological experiments.

We have simulated 11 times the LSMs of both, the 2- and 3-way factor node. For each such trial, each input message component $um_i(\cdot)$ was chosen randomly from the interval $[0, 1]$. A trial lasted 100s in simulated time, $m$ was 10,000. Figure 5.9 shows the corresponding results. A strong positive correlation between the readouts’ average firing rates and the percentage of superfluous bits is evident in both cases and the same holds true for the Fano Factor. The firing rates of the 2-way factor are much lower, because a smaller scaling of the factor tensor entries was used. In addition to using the linear part of the squashing function (see argument above), these results provide a further reason for downsampling of the tensor entries should whole graphs be simulated. Figure 5.9 shows that for low firing rates the Poisson assumption is well justified.
Figure 5.9: Verifying the Poisson assumption for the readouts of the 2- and 3-way factor nodes
Top row: a) For each of 11 simulation trials, the spike trains of the 2-way and 3-way factor readouts give rise to empirical distributions $P^*$ of the number of spikes within $W$, for which Fano Factors can be computed (y-axis). Each dot corresponds to one readout subpopulation, whose average firing rate is shown on the x-axis. Small blue dots represent readouts of the 2-way factor and black dots readouts of the 3-way factor. Total data set size therefore is $2 \cdot 11 \cdot 6 = 132$.
b) Shown is the fraction of superfluous bits due to nonoptimal coding, when the empirical distributions $P^*$ are compared to Poisson distributions $P$ of matched rate. Each dot corresponds to one readout subpopulation, whose average firing rate is shown on the x-axis. The y-axis gives the percentage of superfluous bits $D_{\text{norm}}(P^* \mid P)$. The three histograms show example distributions $P$ (red bars) and $P^*$ (green bars) of selected readouts (origins of grey lines). Bottom row: Magnified plots of the results for the 2-way factor. The data points correspond to the blue dots in the top row.

5.3 Chapter Summary & Discussion

We have presented concrete neural implementations of the two abstract, event-based Belief-Propagation (BP) processors of chapter 3. We have first shown how the framework of analog Echo-State Networks (ESNs) can be combined with escape rate models of the Integrate-And-Fire type, to emulate the interspike-interval (ISI) processor for the computation of single output messages in graphical models containing continuous variables. We then verified that the information contained in such a message can be picked up by another ESN that represents a neighboring factor node. In context of the general event-based processor of section 3.3, we have shown how (spiking) Liquid-State Machines (LSMs), combined
with a neurally inspired model of diffusive noise, can be utilized for computing BP messages in graphical models with general discrete variables. Under the constraint of low firing rates these messages preserve the chosen event-coding scheme of the input messages, which justifies the assumption that neighboring factor nodes can indeed pick up the information contained in the former. However, due to time constraints we have not yet evaluated empirically whether the proposed architecture performs well in larger networks that contain multiple interconnected LSMs. In this context the yet unresolved issue of message normalization might also gain importance, since in practice unnormalized messages often converge to zero or infinity [90].

Two different reservoir frameworks (ESNs and LSMs) have been used for the two processors, because with respect to spike-based implementations their respective level of difficulty is different. Many learning tasks are easier to set up with ESNs, that is, to define and tune the network parameters [94]. For an extended period of time we have also tried to apply a LSM with a single liquid-pool to the ISI processor task, however we were unable to achieve acceptable performance in this case. Moreover, we have examined a spiking version of the ‘Evolino’ method [149], but it resulted in only minor improvements of the cost function after learning (in its original version Evolino uses evolutionary methods to tune the reservoir, whose basic units consist of more complicated Long Short-Term Memory cells rather than spiking neurons [59, 149]). Figure 5.10 shows this by displaying only a small decrease in the NMAE performance measure during evolution, after which the steady-state value remains high (around 0.68).

In contrast, the ESN approach in this chapter was shown to lead to acceptable results. So far, this is the only method found working. A further advantage of this approach compared to the LSM model is the provision of a biological interpretation beyond networks of neurons. This aspect will be discussed in more detail below. However, the mere use of a single, sigmoidal ESN instead of a single, spiking LSM was still not sufficient to achieve acceptable performance. As explained in section 5.1.1, an additional feature had to be present, namely the auxiliary reservoirs $L_X, L_Y$ and readouts $R_X$ and $R_Y$. Since the outputs of the latter are analog, it is also not clear how their spiking counterparts in the LSM framework should look like, but it is possible that the ISI processor can be implemented as well once a suitable spiking representation of $R_X$ and $R_Y$ is found.

We have realized that ordinary linear regression, as used for training $R_X$ and $R_Y$, is impossible to employ for $R_{out}$, although this method is a standard tool for readout training in most ESN/LSM applications [66, 101, 99, 94]. In principle knowing the ISI distribution to be learned permits inferring the hazard, which then leads to the readout’s membrane voltage through inversion of the escape rate model. This voltage might then be considered a suitable target for learning by means of linear regression (see fig. 5.11 for a detailed account of how this voltage can be computed from input spike data). However, empirically this approach was found not to work. The reason is that linear regression minimizes the $L2$-
norm error of the voltage and thus learns the target ISI distribution only indirectly. We found that
even when this error is very small, the induced error in ISI probability after learning can still be large.
Investigating fig.5.11 in a bottom-up fashion reveals why this is the case: The voltage error translates into
error of the hazard (2.23), which then leads to severe error-propagation for the ISI distribution, due to the
integral in 2.23. This effect is more severe for large ISIs for which this integral accumulates more error.
Hence linear regression cannot produce an acceptable training performance and we are thus left with
learning methods that operate directly on the ISI distribution (see appendix D.1). Interestingly, one of
these methods is online in nature and bears similarities to models of Spike-Timing-Dependent-Plasticity,
a synaptic weight modification scheme that was first observed in slice experiments [14] and which has
subsequently triggered many theoretical models of neural learning [55, 84, 128, 65, 109].

The question of biologically plausible learning relates to the question of how the functionality of
model 5.1.1 may be realized in biological neural systems, e.g. by the functional architecture of the
neocortex. The ESN implementation has a straightforward, if cumbersome, neurobiological interpretation
as a network of spiking neurons: Two sets of recurrent reservoir networks with dedicated readout neurons
serve the purpose of ISI decoding and the approximation of the Sum-Product Rule respectively. The
ISI decoder readouts provide local feedback connections to their respective reservoirs to prolong the
memory capacity of the ISI decoder stage. The readout approximating the SPR provides this feedback
to inform the reservoir about the time of the last fired spike, as required by the abstract processor
description (figure 3.3). This interpretation is cumbersome, because the whole arrangement consisting of
several specifically interconnected networks of neurons has to be realized twice –for forward and backward
passing of messages– for each unobserved variable of the underlying Factor Graph.

By abstracting both network tasks (ISI decoding and SPR approximation) however, our processor can
be placed on a more general level that offers an additional interpretation: Nonlinear synaptic dynamics
and linear summation of dendritic subunit outputs work together to realize the complete BP processor
inside the dendritic tree of a single neuron. The resulting voltage trace conveyed to the soma then leads
to the appropriate hazard of firing. Using the backpropagating action potential, the soma in turn informs
the dendritic tree about the time of the last fired spike. Indeed, by modeling the detailed biophysics
of pyramidal neurons, Poirazi et al. [130] have suggested terminal dendrites to act as classical quasi-
independent sigmoidal units, the workhorse of ANN modeling which has also been used in the ESN
approach. The same authors have also extended this model to a two layer network of sigmoidal units
describing the global input/output behavior of a pyramidal neuron [131]. In this case, when the synaptic
activity vector is replaced by the vector of dendritic subunit outputs, the methods of section D.1 can still
be applied for learning the weights of the output layer. Although in the study of [131] the neurons firing
rate was taken as the relevant variable, the idea of abstracting single neuron processing to multilayer
ANNs has also been examined using the temporal distance between spikes belonging to different inputs \cite{172}. However, the model in \cite{172} may principally be formulated also in terms of temporal distances between spikes of the same input, leading to ISI decoder units. ISI decoding might also be realized, or at least supported, by short term synaptic plasticity mechanisms \cite{167, 166, 138} such as facilitation and depression \cite{68, 107, 75}, features that are present at most central synapses \cite{167}. Therefore, the approach of \cite{172} opens the possibility of 'dendritic reservoirs', so that the ISI-processor might be realized within that framework.

The general event-based processor of section \ref{sec:3.3} in contrast does not seem to fit elegantly to such a style of single neuron processing. It can neither be associated with short-term plasticity mechanisms, nor does it provide meaning to the backpropagating action potential (the role of the reference spike cannot be played by the neurons own spikes, due to thus induced positive feedback, which leads to steadily increasing firing rates). Hence, as an interpretation of information processing in the neocortex, this approach seems to be confined to networks of spiking neurons. Moreover, the general event-based processor is computationally limited, since it can only emulate Belief-Propagation on discrete variables. However, it still possesses some merits on its own. First of all, it is very fault-tolerant to residual errors after learning, which becomes strikingly clear, e.g. when comparing figures \ref{fig:5.6} and \ref{fig:5.7}. As explained before, this is because the processor’s output only needs to match the correct mean instantaneous firing rates and not their precise time courses. Also, the approach can be implemented with a single reservoir only, no additional machinery e.g. for ISI decoding is required. This same reservoir may also be used for computing all output messages of the factor node that is represented by the network. We have shown this by allowing all message-carrying spike trains to enter the reservoir during readout training. Therefore we could have equally trained the readout of any other output message. This stands in marked contrast to the reservoir implementation of the ISI processor, where the unused BP message was deliberately prevented from entering (see fig.\ref{fig:5.1}). Although possible in principle, when all messages are allowed to enter the reservoir, we expect learning to be much harder, because in this case any message-readout not only needs to ignore one of the input messages, but also the feedback spikes of the other message-readouts. This is more difficult to achieve the higher the degree of the factor node, because then the reservoir state vector becomes more and more dependent on information that is useless for any considered readout.

In section \ref{sec:5.2.2} we have considered the disadvantage of squashing the SPR-result to obtain output message $u_{\text{out}}(z)$. However, under specific circumstances squashing might also turn out to be beneficial. Imagine a situation where the firing rate of one of the readouts is too large, whereas the others assume their correct values (i.e. their rate is proportional to the SPR-result). Squashing then diminishes the total message error by 'chopping off' the erroneous message component. This way one source of error (squashing) can counterbalance another one (e.g. noise induced by improper learning of the readouts).
Whether the total effect of squashing in a full graph of interacting nodes is beneficial or not is a difficult question a priori. We assume that its answer depends on various details of the graph, for example the graph-structure, factor node parameterizations and noise levels of the individual message components. Due to time constraints however we have not yet empirically evaluated the interaction of nodes in whole graphs. As mentioned earlier in this thesis, the methods of Ihler et al. [63] may be consulted for error estimation if imprecise messages are used.

In case of squashing being the dominant source of error, we have already shown a possible solution, namely the downscaling of the factor tables to obtain low firing rates. Low rates ($\sim 10$Hz) seem to be the most prominent operating regime of the cortex, rarely exceeding 30Hz. This is true for both, spontaneous activity during sleep [116, 23, 170] and wakefulness [23, 170] and in the latter case even during behavioral tasks [116, 50, 108, 114, 170]. Interestingly the functionality of our event-based BP processor necessitates this operating regime for two reasons: 

a) Minimizing BP message error due to squashing by [3.31]

b) Consistent event-coding of in- and output messages, when Poisson spike trains carry the events (fig.5.9).

Of course there is a plethora of alternative explanations for the presence of low firing rates in cortex, ranging from sparse-coding [115] to economic energy consumption [86, 8]. Nevertheless (a) and (b) are functional reasons that are not purely epiphenomenal.
Figure 5.10: Sculpturing a microcircuit towards computing the SPR by means of the Evolino algorithm [149]. Shown are results of a linear readout approximating the target membrane voltage of an escape-rate neuron, whose output ISI distribution follows output message $m_Z(z) = \int_D \int_D f(x, y, z) \cdot m_X(x) \cdot m_Y(y) \, dx \, dy$ of a 3-way factor node $f(x, y, z) := \| (x - z) \|^2$. The readout receives input from a recurrent network of spiking neurons whose connectivity structure is adapted by an evolutionary algorithm (consisting of selection, uniform crossover and mutation). Starting from random, LSM type of network connectivity, the plots show the normalized mean absolute error (NMAE = y-axis) with progression of evolution (number of evolved generations = x-axis), for the training, validation and testing sets respectively. The NMAE is defined as: $\text{NMAE} = \frac{\langle |V_m - V_{m,\text{tar}}|\rangle}{\langle |V_{m,\text{tar}}|\rangle}$, where $V_m$ is the actual membrane voltage of the readout and $V_{m,\text{tar}}$ the corresponding target voltage (see fig.5.11 for how $V_{m,\text{tar}}$ is computed from the input spike trains). $\langle \cdot \rangle$ denotes the mean across training/validation/testing examples. The training set is the set of data the readout is trained on by means of linear regression. Subsequently, such a trained readout is applied to the validation set, and the corresponding performance is used as an evolutionary fitness measure for selection. Finally, the readout is tested on a separate testing set whose performance results do not enter the learning process in any way. ’Best Network So Far’ gives the performance of the best network so far encountered during evolution with respect to the the testing set.
Figure 5.11: Derivation of a suitable target input membrane voltage/current for training readout neuron $R_{out}$ in fig. 5.1 by linear regression.

Shown is an abstract step-by-step description of how the target mean current (or membrane voltage) can be derived from the ISI statistics of the input spike trains ($m_X, m_Y$ in 5.1). From the input spike trains, the function and integral function spikes can be computed which then lead to the target ISI distribution by summation through a temporal sliding window (eq. 3.5). Renewal theory then relates the ISI distribution to the hazard (eq. 2.24), which in turn permits inferring the membrane voltage according to the chosen escape-rate model (e.g., equation 2.22). This voltage might then be used directly as target for learning by means of linear regression. For that, the peak heights (weights) of the postsynaptic potentials are adjusted during learning. Alternatively, the characteristics of the leaky-integrate-and-fire model (equation 2.19) can be used to deduce the input current that gives rise to the desired membrane voltage, and which can be a suitable target for learning as well. In this case the weights of postsynaptic currents would be adjusted by linear regression.
Chapter 6

Discussion

In this thesis we have proposed several models, placed at various levels of abstraction, for how the Belief-Propagation (BP) algorithm can be emulated using spike-based principles. We have discussed in detail the diverse dis/advantages of these models compared to existing approaches in the individual 'Chapter Summary & Discussion' chapters. Also their relation to and implications for neurobiology have already been discussed there. Here we summarize the most important of these results with respect to the thesis’ objectives and describe possible future work.

6.1 Thesis Summary and Conclusion

Chapter 3 has dealt with abstract, functional approaches, i.e. approaches that are spike-based but entirely disregard the use of neural biophysics. In this context, two processor models have been developed, which both utilize the same underlying meta-principle for mapping BPs fundamental equation (the Sum-Product rule) onto spike-based processing. The core idea behind that principle is to exploit the mathematical equivalence between the Sum-Product rule and the computation of expected values of factor functions. These expected values in turn can be approximated through a simple form of Markov-Chain-Monte-Carlo sampling. In the first model, random samples are given by individual spikes, whose preceding interspike interval (ISI) provides the numerical value of the associated sample. Simulation results have revealed the model to be in excellent agreement with numerical evaluations of the BP algorithm. The second model is slightly more general by principally allowing for a multitude of definitions of spike-based, binary random samples. Again, we have verified this model through simulation, for the specific case when a sample is given by the presence of at least one spike in a predefined time window. Both models offer a great deal of flexibility, because they (a) are functional and hence are not tied to a specific implementation using approximate biophysics and (b) can be applied to any type of existing graphical model (Bayesian
Networks, Markov Random Fields and (Forney) Factor Graphs). Therefore, these approaches may remain relevant, even if their concrete implementations -which we discuss in more detail below- are falsified by experimental research. Moreover, the ISI processor model even permits the representation of graphical models with analog variables, a clear advantage compared to previous spike-based BP models and classical realizations of BP on digital computers.

Concerning the second objective of this thesis, the provision of concrete, biophysical implementations of the abstract BP models, we can only give a rough answer. We have shown in chapter 5 that an Echo-State network, which (a) consists of sigmoidal units in the reservoir and a noisy leaky-integrate-and-fire (IAF) point neuron as readout and (b) is endowed with ISI detection mechanisms and readout feedback, can be trained to approximate the ISI processor. Apart from the straightforward interpretation of this setup as a network of neurons, we have also described in what way it might be regarded as a nonlinear, dendritic filtering stage (reservoir) followed by a somatic linear readout (IAF neuron). Based on biophysical models of dendritic processing described in the literature [130, 131], the rate-based behavior of dendritic subunits can indeed be approximated by sigmoidal units, which hence provides a reasonable interpretation for the reservoir. Readout feedback in turn, which is necessary for a proper match of the abstract ISI processor, can be interpreted as a neuron’s backpropagating action potential. So far however a couple of questions concerning the this approach are not sufficiently answered: (1) The exact biophysical realization of dendritic sigmoidal units whose input consist of temporal distances between spikes. (2) Since reservoir computing necessitates recurrent interactions between units, can the morphology of realistic dendritic trees indeed support the idea of ‘dendritic reservoirs’? (3) In what way can short-term plasticity mechanisms support ISI detection and (4) how can the readout weights be learned in a fully unsupervised manner? Although concerning question (4) we have described an exciting similarity between spike-timing-dependent-plasticity and a stochastic gradient descent procedure on an ISI-related cost function, such learning is not fully unsupervised. Furthermore in this context the relationship between synaptic and dendritic subunit weights also needs to be elucidated.

By pushing forward the interpretation of patchy, horizontal connections in cortex as physical realization of graphical models, objective 3 is concerned with implementations of BP in spiking neural networks. In chapter 4 we have presented a network architecture allowing for implementations of general message-passing algorithms in networks of spiking neurons. This architecture consists of an ensemble of interconnected Liquid-State Machines (LSMs), such that each factor node of a Forney Factor Graph is represented by an individual reservoir population of neurons, and each combination of reservoir and readout implements a specific message emitted by the factor node. The presented method is very flexible, because it allows different formulations of BP using various spike-coding strategies for the messages. In this context we have used a population-rate approach to implement computations of single messages as
well as inference in full graphs (sections 4.2.2 and 4.2.3). Moreover, within the sampling framework of section 3.3, it has been shown that single messages of the above spike-count processor can be computed using the LSM architecture.

Anatomically our LSM model predicts the existence of projection neurons (readouts) whose synapses are highly plastic, since in the reservoir computing framework they are subject to learning. The hypothesized reservoir neurons in contrast should have synapses that change slowly at most. Inside the reservoir the comparatively static, recurrent connectivity these neurons give rise to can either be random as in the standard LSM approach, or structured, if the existence of genetically prewired circuitry is assumed. In the first case the individual reservoirs could serve general purpose computations, i.e. apart from their role in BP message-passing they might also be used e.g. for dynamically switching the graph structure through thalamocortical signals (see section 4.1). In the second case one would assume rather stereotyped reservoir connectivity across cortical areas. We have mentioned in chapter discussion 4.3 neurodevelopmental experiments that support this idea and also theoretical frameworks that may be used to automatically construct such circuits in simulations.

6.2 Outlook & Future Work

The material presented in this thesis triggers several aspects for future work. From a neurobiological point of view, the most important issue might be to define core principles of the proposed models which can be verified or falsified by experimental research.

In case of the abstract ISI processor it seems reasonable to first operate directly on experimental spike trains and examine, whether or not these trains are at odds with the renewal assumption, since the functioning of the processor necessitates spike trains to follow general renewal processes. As discussed earlier, regular spiking neurons in cortex show weak correlation between an ISI and its immediate neighboring ISI. Therefore, it needs to be investigated if the ISI processor may still work in presence of such weak correlations. If the answer is ‘yes’ then one can try to define factor nodes whose output messages reproduce the experimental spike trains. In this context a suitable ISI encoding function has to be found also. For example, we have seen in section 3.2.2 that performing BP message-passing in the log(ISI) domain changes the corresponding ISI distribution quite fundamentally. Hence, the combined effect of the factor function and the ISI encoding function should reproduce the experimental data. Subsequently in this case it would also make sense to solve problems 1-4 above, which deal with the concrete biophysical implementation and learning of the abstract ISI processor. On the other hand, if the approach does not work in presence of ISI correlations, there is little hope for the hypothesis that cortical processing can be captured by the functionality of this processor, regardless of its specific biophysical implementation. The
sampling-based meta-principle for BP however (section 3.1) remains unaffected in this case and might still be used as a starting point for spawning new, concrete neural BP models.

Concerning our LSM architecture for patchy horizontal connectivity (section 4.1), future experiments could assess if there indeed exists a dichotomy consisting of neurons that have highly plastic input synapses (readouts) and neurons with rather static such synapses (reservoir neurons). Although the presence of such a dichotomy is a necessary feature of any LSM model, our approach specifically predicts that it should be spread uniformly across the entire cortex, reflecting the graph-like arrangement of interconnected microcircuits. If the dichotomy exists, the next question to be tackled is whether the connectivity the reservoir neurons (or better: the neurons with static synapses) can be reasonably described as random or not. This way one can assess if the hypothesized reservoirs are general-purpose computational engines (random connectivity), or if there is a repeating pattern of connections that might be genetically encoded and supports special-purpose computations in the context of BP (structured connectivity).

Apart from purely neurobiological implications, this thesis has also provided new paradigms for analog computation in general. In particular, the abstract BP approaches we have laid out in chapter 3 might be conceivably implemented in neuromorphic ‘analog Very-Large-Scale-Integrated’ (aVLSI) electronic circuits. In fact, collaboration with Dr. Shi-Chii Liu from our institute has already led to a simulated aVLSI chip, that can perform ISI-based sampling of arbitrary factor functions -that is, one of the core ingredients of the abstract ISI processor (see fig.3.2). Initial results are promising and for the future we have plans to implement the full processor on an aVLSI substrate. In this context one might also think about strategies to tackle the last step towards full algorithmic generality: BP processing with vectorized, analog variables. For that a way has to be found to couple spike trains from different neurons, such that arbitrary, high-dimensional BP-messages can be represented. Since BP in analog graphical models poses severe challenges for conventional, digital computers, we are quite confident that this approach may provide an interesting alternative.
Appendix A

Details of the Abstract Processor Based on Interspike-Intervals

A.1 Derivation of $S(z,t)$ and $S_F(t)$

Here we prove that the stochastic processes $S(z,t)$ and $S_F(t)$ approximate scaled versions of $uSPR(z)$ and $N$ respectively. The individual sections thereby cover different aspects of this proof, starting with the main statement and ending with proofs of detailed assertions that are needed in the process of proving the main statement. In section A.1.1 we derive the expected values of $S(z,t)$ and $S_F(t)$, which turn out to be proportional to $uSPR(z)$ and $N$ respectively. It is then shown in section A.1.2 that, in relative terms, $S(z,t)$ and $S_F(t)$ converge in probability to these expected values, leading to a steady improvement of the approximation in equation 3.5 for increasing $W$. For all derivations we assume that the input ISI distributions $m_X(x)$ and $m_Y(y)$ are both stationary, rendering $S(z,t)$ and $S_F(t)$ stationary as well.

A.1.1 Derivation of $E[S(z)]$ and $E[S_F]$ 

Let $[G(\delta)](t) := H(t) - H(t - W)$ be the impulse-response of filter $G$ constituting the sliding summation window, where $H(t)$ denotes the Heaviside step function. It follows that $\int_0^{T>W} [G(\delta)](t) \, dt = W$. Let $a_z(t) := \sum_{i=1}^{\infty} f_{z,i} \delta(t - t_i)$ be the function spike train caused by the incoming spiking messages $m_X(x)$ and $m_Y(y)$, where $f_{z,i} := f_z(x_i, y_i)$ represents the $i$-th function spike. Given the verbal definition of $S(z,t)$ in section 3.2.1 we can see that $S(z,t) := [G(a_z)](t)$. With plim $(\cdot)$ denoting convergence in probability of $(\cdot)$ and $E[n_{tot}]$ denoting the expected total number of function spikes in $[0,T]$, stationarity of $S(z,t)$
leads to the following expression for the expected value $\mathbb{E}[S(z)]$:

\[
\mathbb{E}[S(z)] = \text{plim}_{T \to \infty} \frac{1}{T} \int_0^T S(z, t) \, dt
\]  
(A.1)

\[= \text{plim}_{T \to \infty} \frac{1}{T} \cdot W \cdot \sum_{i, t_i \in [0, T]} f_{z,i}
\]  
\[= W \cdot \text{plim}_{T \to \infty} \frac{1}{T} \cdot \mathbb{E}[n_{\text{tot}}] \cdot \mathbb{E}[f_z]
\]

The last equation above follows from the convergence behavior of $\sum_{i, t_i \in [0, T]} f_{z,i}$, which is examined in section A.1.2.

Clearly it holds true that

\[\mathbb{E}[n_{\text{tot}}] = r_{\text{tot}} \cdot T
\]  
(A.2)

with $r_{\text{tot}} := \frac{1}{\mathbb{E}[X]} + \frac{1}{\mathbb{E}[Y]}$ the total rate of incoming spikes along $X$ and $Y$.

Since $f_z(x, y)$ is a deterministic function of ISI random variables $X$ and $Y$, its expected value $\mathbb{E}[f_z]$ is given by [124]:

\[
\mathbb{E}[f_z] = \int_D \int_D f_z(x, y) p_s(x, y) \, dx \, dy
\]  
\[= \int_D \int_D f_z(x, y) m_X(x) m_Y(y) \, dx \, dy
\]  
\[= u_{\text{SPR}}(z)
\]  
(A.3)

where $p_s$ denotes the density of the positional sample vector $s := (x, y)$. In figure A.1 it is shown that, by following the sampling scheme of figure 3.2b, $p_s$ is indeed given by $p_s(x, y) = m_X(x) m_Y(y)$. Plugging A.2 and A.3 into the last expression of A.1 one ends up with:

\[\mathbb{E}[S(z)] = r_{\text{tot}} \cdot W \cdot u_{\text{SPR}}(z)
\]  
(A.4)

The derivation of $\mathbb{E}[S_F]$ runs along the same lines yielding:

\[\mathbb{E}[S_F] = r_{\text{tot}} \cdot W \cdot N
\]  
(A.5)
A.1. DERIVATION OF $S(Z,T)$ AND $S_F(T)$

A.1.2 Convergence Behavior of $S(z)$ and $S_F$

Here we examine the behavior of sums of (integral) function spikes in the limit of large temporal summation window sizes $T$. First, it is proven in the main proposition A.1.2.1 that, under weak conditions imposed on $m_X, m_Y$ and $f$, these sums converge in probability to their respective expected values. Thereby equation 3.5 and the final step of equation A.1 are justified. The proof is based on a decoupling of the limits $T \to \infty$ and $n_f \to \infty$, where $n_f$ is a fixed number of (integral) function spikes disregarding time window $[0, T]$. Accordingly, the claim about limit $n_f \to \infty$ needed by the main proposition is proven in lemma A.1.2.2. In that proof we also specify in detail the above conditions on $m_X$ and $m_Y$. Finally, lemma A.1.2.3 establishes the link between the two types of limits. To simplify notation we refer in the following to function $f$ as both, $f_z$ in case of function spikes and $F$ in case of integral function spikes.

Proposition A.1.2.1. Let $A := \{ i \in \mathbb{N} | t_i \in [0, T] \}$ be a sorted set of indices of (integral) function spikes within some time window $T$, such that $t_i < t_{i+1} \forall i \in A$. Let $b := \sum_{i \in A} f_i$ be the corresponding sum of $n_{\text{tot}} := |A|$ (integral) function spikes. Then, provided that $f$ is bounded and the ISI distributions $m_X$ and $m_Y$ are well behaved (see proof of lemma A.1.2.2 for detailed conditions),

$$\lim_{T \to \infty} \frac{b}{E[n_{\text{tot}}] \cdot E[f]} = 1$$

(A.6)

where $E[f]$ is the expected value of an (integral) function spike when the sampling scheme of figure 3.2b is followed and $\lim_{T \to \infty} C_T = C$ denotes convergence in probability of $C_T$ to $C$ as $T \to \infty$. In other words, as $T \to \infty$ the relative error between $b$ and $E[n_{\text{tot}}] \cdot E[f]$ converges to zero in probability.

Proof. The proof is complicated by the fact that $n_{\text{tot}}$ and the values of the samples $f_i$ are not independent, since the latter are determined through ISIs in the fixed time window $[0, T]$. Therefore, we first decouple the (integral) function spikes from that window by forming $b^* := \sum_{i=1}^{n_f} f_i$, with a fixed number $n_f$, for which it is shown in lemma A.1.2.2 that

$$\lim_{n_f \to \infty} \text{CV}^2[b^*] = 0$$

(A.7)

with $\text{CV}^2[b^*] := \frac{\text{Var}[b^*]}{E[b^*]^2}$ and $E[b^*] = n_f \cdot E[f]$. 


That is, the limit \( \lim_{n_f \to \infty} (\cdot) \) keeps on adding (integral) function spikes to \( b^* \), disregarding whether or not these \( n_f \) number of spikes have occurred in \([0, T]\). It then follows from A.7 that \( \frac{b^*}{n_f \mathbb{E}[f]} \) converges to 1 in mean-square and hence also in probability [124], that is
\[
\text{plim}_{n_f \to \infty} \frac{b^*}{n_f \mathbb{E}[f]} = 1 \quad \text{(A.8)}
\]
holds true. Finally lemma A.1.2.3 shows that assertion A.6 follows from assertion A.8 which completes the proof.

\[\square\]

**Lemma A.1.2.2.** Let \( b^* := \sum_{i=1}^{n_f} f_i \) be a sum of \( n_f \) (integral) function spikes that are not restricted to fall into any temporal summation window. Define \( \text{CV}^2[b^*] := \frac{\text{Var}[b^*]}{\mathbb{E}[b^*]^2} \) as the squared coefficient of variation of \( b^* \). Then, provided that \( f \) is bounded and the ISI distributions \( m_X \) and \( m_Y \) are well behaved,
\[
\lim_{n_f \to \infty} \text{CV}^2[b^*] = 0 \quad \text{(A.9)}
\]
and
\[
\mathbb{E}[b^*] = n_f \cdot \mathbb{E}[f] \quad . \quad \text{(A.10)}
\]

**Proof.** For the proof we need to derive \( \mathbb{E}[b^*] = n_f \cdot \mathbb{E}[f] \) and an upper bound on \( \text{Var}[b^*] \) that increases slower than \( \mathbb{E}^2[b^*] \) for \( n_f \to \infty \). The former is easily achieved by using the result of figure A.1 from which it follows that the arguments \((x_i, y_i)\) of each (integral) function spike \( f_i(x_i, y_i) \) are identically distributed. Therefore, the expected value of the sum of \( n_f \) such spikes is given by \( n_f \cdot \mathbb{E}[f] \).

To achieve the latter we first write
\[
\text{Var}[b^*] = \sum_{i=1, j=1}^{n_f} \text{Cov}[f_i, f_j] = n_f \cdot \text{Var}[f] + 2 \sum_{i=1}^{n_f} \sum_{i<j \leq n_f} \text{Cov}[f_i, f_j] \quad \text{(A.11)}
\]
Note that \( f_i \) and \( f_j \) are independent, if at least one \( X \)- and one \( Y \)-spike have occurred between \( f_i \) and (including) \( f_j \). This is due to the renewal property and mutual independence of the two spike trains, which assure that both arguments \( X, Y \) of \( f \) have been independently updated in this case. Accordingly, we define by \( \xi_k \in \{0, 1\} \) the binary event that such a two-fold update has happened for two (integral) function spikes with index difference \( k = j - i \). Therefore, \( \text{Cov}[f_i, f_j \mid \xi_{j-i} = 1] = 0 \). With \( P_{\xi_k}(\xi_k = 1/0) \)
denoting the probability of $\xi_k$, we may use the law of total covariance to decompose $\text{Cov} [f_i, f_j]$ for $i < j$:

$$\text{Cov} [f_i, f_j] = \mathbb{E} \left[ \text{Cov} [f_i, f_j \mid \xi_{j-1}] \right] + \mathbb{E} \left[ \text{Cov} [f_i \mid \xi_{j-1}], \mathbb{E} [f_j \mid \xi_{j-1}] \right]$$  \hspace{1cm} (A.12)

$$\mathbb{E} \left[ \text{Cov} [f_i, f_j \mid \xi_{j-1}] \right] = \text{Cov} [f_i, f_j \mid \xi_{j-1} = 0] \cdot P_{\xi_{j-1}}(\xi_{j-1} = 0)$$  \hspace{1cm} (A.13)

$$+ \text{Cov} [f_i, f_j \mid \xi_{j-1} = 1] \cdot P_{\xi_{j-1}}(\xi_{j-1} = 1)$$

$$= \text{Cov} [f_i, f_j \mid \xi_{j-1} = 0] \cdot P_{\xi_{j-1}}(\xi_{j-1} = 0)$$

$$\text{Cov} \left[ \mathbb{E} [f_i \mid \xi_{j-1}], \mathbb{E} [f_j \mid \xi_{j-1}] \right] = \mathbb{E} \left[ \mathbb{E} [f_i \mid \xi_{j-1}] \cdot \mathbb{E} [f_j \mid \xi_{j-1}] \right]$$  \hspace{1cm} (A.14)

$$- \mathbb{E} \left[ \mathbb{E} [f_i \mid \xi_{j-1}] \right] \cdot \mathbb{E} \left[ \mathbb{E} [f_j \mid \xi_{j-1}] \right]$$

$$= \mathbb{E} [f_i \mid \xi_{j-1} = 0] \cdot \mathbb{E} [f_j \mid \xi_{j-1} = 0] \cdot P_{\xi_{j-1}}(\xi_{j-1} = 0)$$

$$+ \mathbb{E} [f_i \mid \xi_{j-1} = 1] \cdot \mathbb{E} [f_j \mid \xi_{j-1} = 1] \cdot P_{\xi_{j-1}}(\xi_{j-1} = 1)$$

$$- \mathbb{E} [f_j]$$

We can infer from figure A.1 that $f_i$ and $\xi_{j-1}$ are independent, i.e. that $\mathbb{E} [f_i \mid \xi_{j-1}] = \mathbb{E} [f_i]$. However, in general the same must not hold for $f_j, i < j$. Hence it follows from A.14:

$$\text{Cov} \left[ \mathbb{E} [f_i \mid \xi_{j-1}], \mathbb{E} [f_j \mid \xi_{j-1}] \right] = \mathbb{E} [f_i] \left( \mathbb{E} [f_j \mid \xi_{j-1} = 0] \cdot P_{\xi_{j-1}}(\xi_{j-1} = 0) \right.$$

$$+ \mathbb{E} [f_j \mid \xi_{j-1} = 1] \cdot P_{\xi_{j-1}}(\xi_{j-1} = 1) \right) - \mathbb{E} [f_j]^2$$

$$= \mathbb{E} [f_j] \left( \mathbb{E} [f_j \mid \xi_{j-1}] \right) - \mathbb{E} [f_j]^2 = 0$$

Since $f \geq 0$, plugging A.13 and A.15 into A.12 yields:

$$\text{Cov} [f_i, f_j] = \text{Cov} [f_i, f_j \mid \xi_{j-1} = 0] \cdot P_{\xi_{j-1}}(\xi_{j-1} = 0)$$  \hspace{1cm} (A.16)

$$\leq M^2 \cdot P_{\xi_{j-1}}(\xi_{j-1} = 0)$$  \hspace{1cm} (A.17)

with $M := \sup [f]$. Therefore,

$$\text{Var} [b^\ast] \leq n_f \text{Var} f + 2M^2 \sum_{i=1}^{n_f} \sum_{i < j \leq n_f} P_{\xi_{j-1}}(\xi_{j-1} = 0)$$  \hspace{1cm} (A.18)

We now assume that $m_X(x), m_Y(y)$ are well behaved, i.e. that

$$\lim_{n_f \to \infty} \sum_{i < j \leq n_f} P_{\xi_{j-1}}(\xi_{j-1} = 0) = C,$$ with $C$ some constant. For biologically relevant ISI distributions this condition can always be fulfilled by assuming arbitrarily small/large minimum/maximum ISIs, below/above which the two densities vanish. In this case a two-fold coordinate update ($\xi_k = 1$) will always
happen after a finite number $K$ of updates, i.e. $P_{i-1}(\xi_k = 0) = 0$, $\forall k \geq K$. Hence,

$$\text{Var}[b^*] \leq n_f (\text{Var}[f] + 2M^2C)$$

(A.19)

\[\square\]

Figure A.1: Density of positional sample vectors and (in)dependencies between spike-based quantities

- **a)** Four basic independency statements: 1) Without loss of generality suppose the last (integral) function spike $f_i$ to have happened at time $t_i$ along $X$. $f_i$ depends exclusively on the positional sample vector $s_i := (x_i, y_i)$, given by the last two respective ISIs along $X$ and $Y$. In other words, given $s_i$, $f_i$ is conditionally independent of any other variable. 2) Denote by $\Delta t_i$ the difference between $t_i$ and the time of the last spike along $Y$ before $t_i$, i.e. $\Delta t_i = t_i - t_{i-1}$ in the illustrated case. Denote by $S^*_i$ the state vector $S^*_i := (t_i, \Delta t_i)$. Because the two spike trains are independent and renewal, $S^*_i$ is not informative about the ISIs $x_i, y_i$ before $t_i$ and $t_j$ respectively. It follows $p_s(s_i | S^*_i) = m_X(x_i)m_Y(y_i) = p_s(s_i)$, in particular $s_i$ and $S^*_i$ are independent. 3) With $l_i \in \{0, 1\}$ denoting a binary label indicating whether or not spike $i$ was received along $X$, it also follows from the renewal property and spike train independence that, given $S^*_i$, future spiking behavior $t^*_i := (t_{i+1}, l_{i+1}, t_{i+2}, l_{i+2}, ...)$ is conditionally independent from any other variable and hence also from past spiking $(t_{i-2}, l_{i-2}, t_{i-3}, l_{i-3}, ...)$. In particular $p_S(t^*_i | S^*_i, s_i) = p_S(t^*_i | S^*_i)$. 4) $\xi_{j-i}, j > i$ is a function exclusively of $t^*_i$, i.e. $\xi_{j-i}$ is conditionally independent from any other variable.

- **b)** Bayesian network modeling the (in)dependencies between the variables in (a). Any additional edge would violate any of the four independence statements (1), (2), (3), (4) in (a). Clearly, it follows from the graph that $f_i$ and $\xi_{j-i}$ are marginally independent, that is $P_{i-1}(\xi_{j-i} | f_i) = P_{i-1}(\xi_{j-i})$. 
It remains to be shown in proposition A.1.2.1 that assertion A.6 follows from assertion A.8, which is achieved by the following lemma.

**Lemma A.1.2.3.** Let all quantities be as defined in proposition A.1.2.1. If

$$\lim_{n \to \infty} b^*_{n_f} \cdot \mathbb{E}[f] = 1$$

(A.20)

then

$$\lim_{T \to \infty} \frac{b}{\mathbb{E}[n_{tot}] \cdot \mathbb{E}[f]} = 1$$

(A.21)

with \(\mathbb{E}[n_{tot}] = r_{tot} \cdot T\) and \(r_{tot} := \frac{1}{\mathbb{E}[X]} + \frac{1}{\mathbb{E}[Y]}\) denoting the total rate of spikes coming in as messages along X and Y respectively.

**Proof.** Preliminaries: To avoid cluttered notation, define \(n := n_f\) and let \(p(x)\) be the shorthand for the probability density \(p_X(x)\) of some random variable \(X\) evaluated at \(x\). Disregarding window \([0, T]\), define by \(T_{\epsilon,n}\) the time such that after \(n\) (integral) function spikes the probability of \(t_n \geq T_{\epsilon,n}\) is \(\epsilon\), with \(\epsilon > 0\) an arbitrary constant. Denote by \(b^*_{n}\) the random variable that is equal to \(b^*\) in case of \(t_n < T_{\epsilon,n}\) and hence follows the same distribution as \(p(b^* | t_n < T_{\epsilon,n})\). Analogously, \(b_n\) denotes the random variable that is equal to \(b\) in case of \(t_n < T_{\epsilon,n} \leq T\). The difference between \(b^*_{n}\) and \(b_n\) is that the latter might comprise additional (integral) function spikes in \([t_n, T]\), i.e. \(n_{tot} > n\) in this case. Let \(m(\epsilon, T)\) be the largest (integral) function spike index such that \(T_{\epsilon,m(\epsilon,T)} \leq T\) holds for any given \(T\), i.e. \(m(\epsilon, T) := \max(n | T_{\epsilon,n} \leq T)\).

Because the (integral) function spikes are governed by independent renewal processes, we know that

$$\lim_{n \to \infty} \frac{n}{r_{tot} \cdot t_n} = 1.$$  

It follows that if (A.20) holds, then

$$\lim_{n \to \infty} \frac{b^*}{r_{tot} \cdot t_n \cdot \mathbb{E}[f]} = 1$$

(A.22)

since \(\lim_{n \to \infty} (C_n \cdot D_n) = \lim_{n \to \infty} C_n \cdot \lim_{n \to \infty} D_n\) holds for convergent sequences of random variables \(C_n, D_n\).

Furthermore, by the above definitions we know that:

$$p\left(\frac{b^*}{r_{tot} \cdot t_n \cdot \mathbb{E}[f]} \middle| t_n < T_{\epsilon,n}\right) (1 - \epsilon)
+ p\left(\frac{b^*}{r_{tot} \cdot t_n \cdot \mathbb{E}[f]} \middle| t_n \geq T_{\epsilon,n}\right) \epsilon$$

(A.23)
and
\[
p \left( \frac{b^*}{r_{tot} \cdot t_n \cdot E[f]} \left| t_n < T_{\epsilon,n} \right. \right) = p \left( \frac{b_n^*}{r_{tot} \cdot t_n \cdot E[f]} \right) \tag{A.24}
\]

Hence, if \( A.22 \) holds then
\[
\forall \epsilon_1, \delta > 0 \exists N : P \left( \frac{b^*}{r_{tot} \cdot t_n \cdot E[f]} - 1 > \delta \right) < \epsilon_1, \forall n \geq N \tag{A.25}
\]
\[
\Rightarrow P \left( \frac{b_n^*}{r_{tot} \cdot t_n \cdot E[f]} - 1 > \delta \right) < \frac{\epsilon_1}{1 - \epsilon} \tag{A.26}
\]
\[
\Rightarrow \lim_{n \to \infty} \frac{b_n^*}{r_{tot} \cdot t_n \cdot E[f]} = 1 \tag{A.27}
\]

where \( P(\alpha) \) is the probability of event \( \alpha \). Because \( m(\epsilon, T) \) is monotonically increasing as a function of \( T \), we can conclude from \( A.27 \) that
\[
\lim_{T \to \infty} \frac{b_{m(\epsilon,T)}^*}{r_{tot} \cdot T \cdot E[f]} = 1 \tag{A.28}
\]

Additionally, since the (integral) function spikes are governed by independent renewal processes, we know that \( \lim_{T \to \infty} \frac{t_{m(\epsilon,T)}}{T} = 1 \), from which \( \lim_{T \to \infty} \frac{b_{m(\epsilon,T)}}{r_{tot} \cdot T \cdot E[f]} = 1 \) follows, leading to
\[
\lim_{T \to \infty} \frac{b_{m(\epsilon,T)}}{r_{tot} \cdot T \cdot E[f]} = 1 \tag{A.29}
\]

Using
\[
p \left( \frac{b}{r_{tot} \cdot T \cdot E[f]} \left| t_{m(\epsilon,T)} < T_{\epsilon,m(\epsilon,T)} \right. \right) = (1 - \epsilon) \cdot p \left( \frac{b}{r_{tot} \cdot T \cdot E[f]} \left| t_{m(\epsilon,T)} < T_{\epsilon,m(\epsilon,T)} \right. \right) \tag{A.30}
\]
\[
+ \epsilon \cdot p \left( \frac{b}{r_{tot} \cdot T \cdot E[f]} \left| t_{m(\epsilon,T)} \geq T_{\epsilon,m(\epsilon,T)} \right. \right)
\]

and
\[
p \left( \frac{b}{r_{tot} \cdot T \cdot E[f]} \left| t_{m(\epsilon,T)} < T_{\epsilon,m(\epsilon,T)} \right. \right) = p \left( \frac{b_{m(\epsilon,T)}}{r_{tot} \cdot T \cdot E[f]} \right) \tag{A.31}
\]
we can conclude from \( A.29 \)

\[
\forall \epsilon_1, \delta > 0 \exists T_0 : P \left( \frac{b_{E(f)}}{T \cdot E[f]} - 1 > \delta \right) < \epsilon_1, \forall T \geq T_0 \tag{A.32}
\]

\[
\Rightarrow P \left( \frac{b_{E(f)}}{T \cdot E[f]} - 1 > \delta \right) < (1 - \epsilon_1) \cdot \epsilon_1 + \epsilon_1 \cdot 1 \tag{A.33}
\]

\[
\Rightarrow \lim_{T \to \infty} \frac{b}{T \cdot E[f]} = 1 \tag{A.34}
\]

where the last implication follows from our free choice of \( \epsilon_1 \), e.g. as \( \epsilon < \epsilon_1 \).

\[\square\]

\section{A.2 Parameters Used for the Black-Box Implementations of the Abstract Processor}

The parameters below were used for the simulations in section 3.2.2 (figures 3.5 and 3.6). Quantities based on time are measured in dimensionless steps:

\( W = 750 \); Domain of variable values (possible ISI values) = \( \{ i \in \mathbb{N} \mid i \leq 100 \} \) (if by chance a processor did not fire after at most 200 steps, a spike was forced to keep the inferential dynamics ongoing);

Number of gaussian mixture components defining factor nodes of dimension 1/2/3/4: 5/10/20/20 respectively; Range of standard deviations along principal components of the gaussians defining the mixture of 1/2/3/4-dimensional factors: \([3/7/10]/[3/15]/[3/15] \) respectively (factors of higher dimension needed larger standard deviations and/or more mixture components to maintain a reasonable filling of their domain space); Range of unnormalized weights of the mixture components = \([0.1, 1] \); Span used for moving average filtering of the raw ISI histograms: 3 bins. Each output message was represented by four independent renewal processes whose statistics were all controlled by the same input spike trains.

For the HMM problem of figure 3.10 (inference over time), the following parameters have been used (quantities parameterizing factor nodes are defined in the log(ISI) domain):

\( W = 1500 \); standard deviation along the second and third principal component of the gaussians defining equality constraint factor nodes: 0.01; \( \delta = 0.06; \sigma = 0.03; \sigma_V = 0.06; \sigma_A = 0.1 \); if a processor had ceased to fire for 250 steps a spike was forced. Each output message was represented by 15 independent renewal processes, whose statistics were all controlled by the same input spike trains.
Appendix B

Calculation of $\mathbb{E} [D(B^* \parallel P)]$ and $\text{Var} [D(B^* \parallel P)]$ Used in Figures 3.5, 3.6, 5.2 and 5.3

Given a fixed distribution $P(x)$ we are interested in having expressions for $\mathbb{E} [D(B^* \parallel P)]$ and $\text{Var} [D(B^* \parallel P)]$, when $B^*(x)$ is a random distribution taken from a uniform distribution of distributions of variable $X$. For that we first require the marginal probability densities $P_{\text{single}}(b_x)$ and $P_{\text{pair}}(b_x, b_y)$ of the coefficients $b_x := B^*(x); b_y := B^*(y)$. These densities are necessary to compute the various expected values occurring in the expressions for $\mathbb{E} [D(B^* \parallel P)]$ and $\text{Var} [D(B^* \parallel P)]$. When the distribution $B^*(x)$ is drawn uniformly from the $(n - 1)$-dimensional probability simplex –with $n$ denoting the number of states of $X$– we obtain the following:

$$P_{\text{single}}(b_x) = (n - 1)(1 - b_x)^{n-2} \quad (B.1)$$
$$P_{\text{pair}}(b_x, b_y) = (n - 1)(n - 2)(1 - b_x - b_y)^{n-3} \quad (B.2)$$

Define $f(b_x, p_x) := b_x \cdot \ln \left( \frac{b_x}{p_x} \right)$ and let $\gamma \approx 0.577$ be the Euler-Mascheroni constant. Denote by $H_n$ the $n$-th Harmonic Number and by $\mathcal{U}(n)$ the uniform distribution over $n$ states.
It follows:

\[ \mathbb{E}[D(B^*\|P)] = \sum_x \mathbb{E}[f(b_x,p_x)] \quad (B.3) \]

\[ = \sum_x \int_0^1 P_{\text{single}}(b_x) f(b_x,p_x) \, db_x \quad (B.4) \]

\[ = 1 - H_n - \sum_x \ln p_x \quad (B.5) \]

\[ \approx 1 - \gamma + \sum_x \left( \frac{1}{n} \ln \left( \frac{1}{n} \right) - \frac{1}{n} \ln p_x \right) ; \quad (n \to \infty) \quad (B.6) \]

\[ = (1 - \gamma) + D(U(n)\|P) \quad (B.7) \]

\[ \text{Var}[D(B^*\|P)] = \sum_x \text{Var}[f(b_x,p_x)] + \sum_{x \neq y} \text{Cov}[f(b_x,p_x), f(b_y,p_y)] \quad (B.8) \]

\[ = \sum_x \mathbb{E}[f^2(b_x,p_x)] + \sum_{x \neq y} \mathbb{E}[f(b_x,p_x) \cdot f(b_y,p_y)] - \]

\[ \sum_{x,y} \mathbb{E}[f(b_x,p_x)] \cdot \mathbb{E}[f(b_y,p_y)] \quad (B.9) \]

computation of the integrals

\[ \mathbb{E}[f^2(b_x,p_x)] = \int_0^1 P_{\text{single}}(b_x) f^2(b_x,p_x) \, db_x \quad (B.10) \]

\[ \mathbb{E}[f(b_x,p_x) \cdot f(b_y,p_y)] = \int_0^{1-b_x} \int_0^{1-b_y} P_{\text{pair}}(b_x,b_y) f(b_x,p_x) f(b_y,p_y) \, db_y \, db_x \quad (B.11) \]

and subsequent rearrangement/elimination of terms then yields:

\[ \text{Var}[D(B^*\|P)] = \frac{3n \|l_P\|_2^2 + (\pi^2 - 6) n^2 - 3 \|l_P\|_1^2}{3n^2(n+1)} - \psi^{(1)}(n+1) \quad (B.12) \]

where \( \|l_P\|_1 \) and \( \|l_P\|_2 \) are respectively the 1- and 2-norm of vector \( l_P := \{\ln(p_{x_1}), \ldots, \ln(p_{x_n})\} \) and \( \psi^{(1)}(\cdot) \) is the polygamma function of order 1.
Appendix C

Supplementary Information for the Simulations of Chapter 4

C.1 Details of the LSM Implementations

We have used different parameter settings for the liquid populations of different factor nodes. The parameter $\lambda$ controls the width of the gaussian giving the distance dependent probability of establishing a synaptic connection between two neurons of the liquid population (see [101]). Different values of $\lambda$ were empirically found to be optimal for different factor nodes. Connections from neurons outside to neurons inside the liquid pool were drawn randomly with a fixed, distance independent probability. However those inputs were restricted to certain subparts of the liquid pool. Since the computation of a message in one direction (see equation 2.1 in the paper) requires to ignore the message coming in along the same edge from the other direction, a restriction of the inputs to distinct regions diminished an undesired mixing of information. Values for $\lambda$ and the dimensions of the liquid pools are summarized in Tables C.1 and C.2. The small values of $\lambda$ for liquid pools defining simple factors suggests that in these cases the liquid mainly serves the same purpose as a hidden layer in a multilayer perceptron.

The neural parameters of the 'leaky-integrate-and-fire' neurons in the liquid pools were identical to those in [99], except that the noise current injected at each time step $\Delta t = 0.1$ms was set to zero (see also next section). The 343 neurons in a readout population were also made as similar as possible to the design in [99]. Slightly different were their fixed reset voltage ($V_{\text{reset}} = 10$mV), the random threshold voltage taken uniformly and independently for each neuron from the interval [15 20]mV, the refractory period of 2ms and the constant injected background current which was determined during the training procedure by linear regression. Also different was the noise current injected at each time step which
was set to 161nA in order to implement Poissonian firing. This rather high value is due to the necessity of normalizing the noise amplitude of a white noise process by $\sqrt{\Delta t}$ when the membrane potential is calculated by the Euler method. All of the readout neurons were excitatory.

Synaptic parameters also followed closely the values given in [101, 99], however we did not use dynamic synapses (no facilitation and depression). PSCs were modeled as low-pass filtered spike trains according to: $PSC(t) = W_0 \exp(-\frac{t-t_i-D}{\tau_s})$, where $W_0$ is the instantaneous step(syn. weight) of the postsynaptic current in response to a spike arriving at the synapse at time $t_i$ having an impact after a synaptic transmission delay $D$. $\tau_s$ is the decay time constant. The weights $W$ were drawn from a gamma, the delays $D$ from a normal distribution. Table C.3 gives an overview over the used values. Because the connections from the liquid to the readout populations were determined by linear regression, the associated weights did not obey sign constraints (i.e. also inhibitory liquid neurons could form excitatory synapses and vice versa).

<table>
<thead>
<tr>
<th>Factor</th>
<th>Dimensions of liquid pool</th>
<th>$\lambda$</th>
<th>$C_{scale}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1(S</td>
<td>R,O)$</td>
<td>5<em>5</em>78</td>
<td>2</td>
</tr>
<tr>
<td>$f_\omega(O,O',O'')$</td>
<td>5<em>5</em>42</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td>$P_2(C</td>
<td>O)$</td>
<td>5<em>5</em>42</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table C.1: Parameters of the liquid pools defining the individual factors of the 'Explaining Away' graph

<table>
<thead>
<tr>
<th>Factor</th>
<th>Dimensions of liquid pool</th>
<th>$\lambda$</th>
<th>$C_{scale}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{1...4}(S</td>
<td>R,O)$</td>
<td>5<em>5</em>12</td>
<td>0.7</td>
</tr>
<tr>
<td>$f_\omega(O,O',O'')$</td>
<td>5<em>5</em>40</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td>$f_\phi(O,O',O'')$</td>
<td>5<em>5</em>78</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table C.2: Parameters of the liquid pools defining the individual factors of the noisy channel problem
### C.1. DETAILS OF THE LSM IMPLEMENTATIONS

<table>
<thead>
<tr>
<th>Connection Type</th>
<th>$\mu_W$ (nA)</th>
<th>$CV_W$</th>
<th>$\mu_D$ (ms)</th>
<th>$CV_D$</th>
<th>$\tau_s$ (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inside liquid: EXC→EXC</td>
<td>30</td>
<td>0.7</td>
<td>1.5</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>inside liquid: EXC→INH</td>
<td>60</td>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>inside liquid: INH→EXC</td>
<td>-19</td>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>6</td>
</tr>
<tr>
<td>inside liquid: INH→INH</td>
<td>-19</td>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>6</td>
</tr>
<tr>
<td>readout to liquid: EXC→EXC</td>
<td>4.5</td>
<td>0.7</td>
<td>1.5</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>readout to liquid: EXC→INH</td>
<td>9</td>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>liquid to readout</td>
<td>-</td>
<td>-</td>
<td>1.5</td>
<td>-</td>
<td>6</td>
</tr>
</tbody>
</table>

Table C.3: Synaptic parameters of all connection types. $\mu_W$ and $\mu_D$ indicate the mean weight and synaptic delay respectively, $CV_W$ and $CV_D$ are the corresponding coefficients of variation. These parameters were used to assign random values to each synapse. $\tau_s$ is the synaptic time constant, the '-' indicates learned synapses.
C.2 Learning Performance With Noise Applied to the Liquid-Pool

For the simulations described in the paper we did not apply noise to the neurons in the liquid populations. However figure C.1 shows that learning the Belief-Propagation messages is robust to noise applied to the liquid neurons, provided the noise-level is not larger than $\approx 10\%$ of the dynamic range of the neurons. There is even a small stochastic resonance phenomenon at $2\%$ noise level which was consistent across several repetitions of the whole learning sequence (not shown). The overall performance in figure C.1 is worse compared to the results presented in the paper because much less training data was used.

![Figure C.1: Learning performance of a Liquid-State Machine (LSM) with noise applied to the neurons in the liquid pool. Shown are the results of 12 trials of learning the input current to readout neurons implementing a '='-node (see figure 4.3 and equation 4.3). During each run, a different level of white noise was applied to the total input current of each neuron in the liquid population. The membrane potentials of these neurons therefore could be characterized by Ornstein-Uhlenbeck processes, i.e. by gaussian jitter around a noise-free reference trajectory. The x-axis represents the ratio between the (limiting) standard deviation of that gaussian jitter ($\sigma_{V_m}$) and the dynamic range of the neurons in the liquid ($V_\theta - V_r$). The y-axis shows the signal-to-noise ratio of the output of a trained LSM when exposed to testing data. The signal-to-noise ratio is given by $\frac{\sigma_{target}}{\sigma_{error}}$, where $\sigma_{target}$ is the standard deviation of the target output and $\sigma_{error}$ the standard deviation of the error between the target and the actual output of the LSM. For zero noise (not shown), the performance was the same as for $\sigma_{V_m}/(V_\theta - V_r) = 1\%$.](attachment:image.png)
C.3 Examples of Message Update Equations

Below, two examples of message equations as they were used to compute the target time series are shown. They are obtained by applying equation 4.1 to the corresponding factors defined in section 4.2.3.

- Message $m_{P_1 \rightarrow P_3}$ in Figures 4.6, 4.7, 4.8 ('Explaining Away' problem):

  let $m_1 := m_{S \rightarrow P_1}$, $m_2 := m_{S \rightarrow P_1}$

  \[
  \Rightarrow \tau \dot{m}_{P_1 \rightarrow P_3}(t) + m_{P_1 \rightarrow P_3}(t) = \frac{M(t,1)}{M(t,0) + M(t,1)}
  \]

  where

  \[
  M(t,0) := 0.8(1 - m_1(t))(1 - m_2(t)) + 0.2(1 - m_1(t))m_2(t) + 0.2m_1(t)(1 - m_2(t)) + 0.8m_1(t)m_2(t)
  \]

  \[
  M(t,1) := 0.2(1 - m_1(t))(1 - m_2(t)) + 0.9(1 - m_1(t))m_2(t) + 0.8m_1(t)(1 - m_2(t)) + 0.1m_1(t)m_2(t)
  \]

- Message $m_{\oplus \rightarrow \oplus}$ in figures 4.2, 4.4 and 4.5 ('Noisy Channel' problem):

  let $m_1 := m_{P_1 \rightarrow \oplus}$, $m_2 := m_{P_2 \rightarrow \oplus}$

  \[
  \Rightarrow \tau \dot{m}_{\oplus \rightarrow \oplus}(t) + m_{\oplus \rightarrow \oplus}(t) = \frac{M(t,1)}{M(t,0) + M(t,1)}
  \]

  where $M(t,0) := (1 - m_1(t))(1 - m_2(t)) + m_1(t)m_2(t)$

  \[
  M(t,1) := m_1(t)(1 - m_2(t)) + (1 - m_1(t))m_2(t)
  \]
Appendix D

Supplementary Information for the Simulations of Chapter 5

D.1 Derivation of the Cost Function Used for Training $R_{out}$

Readout $R_{out}$ in figure 5.1 has been trained using a different cost function than readouts $R_X$ and $R_Y$. For a spiking neuron subject to escape noise—with hazard activation function $h(t, w) := h(V_m(t, w))$, whose membrane potential $V_m$ is given by a weighted sum $V_m(t, w) = w^T \cdot X(t)$ of its synaptic inputs $X(t)$—we derive here suitable cost functions for learning the neuron to fire with a given ISI distribution $p(\Delta t, t)$. This distribution is explicitly allowed to depend on $t$ (as for example in equation 3.5). The first learning procedure is based on the log-likelihood of an observed spike train and is semi-supervised and online in nature. In the context of learning precise temporal spike patterns, this method has first and independently been described by [128]. These authors also show how it relates to biologically plausible Spike Time Dependent Plasticity (STDP) weight update schemes. We also present a second related cost that is easier to work with in practice and which has been used to produce the results of section 5.1.2. Denote by $T_s := \{t_i | i = 1...N, S(t_i) = 1\}$, the sorted set of spike times of the learning neuron, with $S(t_i) := 1$ iff the neuron has fired a spike at time $t$, $S(t_i) := 0$ otherwise. As before, let $p(\Delta t, t)$ be the target unconditioned ISI distribution at time $t$, and $q(\Delta t, t, w)$ be the corresponding actual unconditioned ISI distribution of the neuron. For the online method suppose the ISI distribution of $T_s$ follows $p(\Delta t, t)$ and the neuron is teacher-forced to fire at times $T_s$. Then our goal is to minimize the negative log-likelihood $C(w)$ of the training set $T_s$: 
\( C(w) := \sum_{i=2}^{N} -\ln q(t_i - t_{i-1}, t_i, w); \quad t_i \in T_s, \quad N := |T_s| \quad (D.1) \)

Using standard results of renewal theory, \( q(\Delta t, t_{i-1} + \Delta t, w) \) can be explicitly expressed in terms of the hazard as \([28, 55]\):

\[
q(\Delta t, t_{i-1} + \Delta t, w) = h(t_{i-1} + \Delta t, w) \cdot \exp \left( - \int_{0}^{\Delta t} h(t_{i-1} + \Delta t', w) \, dt' \right) \quad (D.2)
\]

yielding

\[
C(w) = \sum_{i=2}^{N} -\ln h(t_i, w) + \int_{t_{i-1}}^{t_i} h(t', w) \, dt' \quad (D.3)
\]

After having computed the gradient of equation \( D.3 \) \([128]\), one can perform a stochastic gradient descent procedure \([19]\) based on a biologically plausible, online learning paradigm. Interestingly, on the postsynaptic side learning in this fashion is feasible only at time points when the neuron spikes, i.e. at times \( t_i \in T_s \), since expression \( D.3 \) requires integration from \( t_{i-1} \) to \( t_i \) \([128]\). This feature however is a necessary requirement in many biological accounts of learning, e.g. in models involving STDP \([128, 55]\). Furthermore, under rather mild and biologically plausible conditions on \( h(V_m) \), \( C(w) \) is a convex function and therefore devoid of local minima \([123]\). These conditions are also fulfilled by \( h(V_m) = h_{\theta} e^{\frac{V_m - V_{\theta \sigma}}{V_{\theta \sigma}}} \), the hazard activation function commonly used in the neuroscientific literature \([128, 127, 136, 37]\) and in section 5.1.1. Despite these desirable properties, the presented optimization method is technically impractical for the processor approximation task of section 5.1.1. Since, as stated before, weight updates are restricted to times when the learning neuron spikes, times \( t' \) between spikes cannot be used for learning, leading to long learning times until enough training examples have been absorbed for proper learning. Hence, the results presented in section 5.1.1 are based on optimizing a slightly different cost function \( C_{\text{batch}}(w) \) in batch-mode using the BFGS quasi-Newton method \([12]\):

\[
C_{\text{batch}}(w) := \sum_{i=2}^{N} D(p_i^* \| q_i^*) \quad (D.4)
\]

\[
p_i^*(\Delta t) := \frac{p(\Delta t, t_{i-1} + \Delta t)}{\int_{0}^{t_i - t_{i-1}} p(\Delta t', t_{i-1} + \Delta t', w) \, dt'} \quad (D.5)
\]

\[
q_i^*(\Delta t, w) := \frac{q(\Delta t, t_{i-1} + \Delta t, w)}{\int_{0}^{t_i - t_{i-1}} q(\Delta t', t_{i-1} + \Delta t', w) \, dt'}; \quad t_i \in T_s \quad (D.6)
\]
where \( p^*_i(\Delta t) \) and \( q^*_i(\Delta t, w) \) are the above target and actual ISI distributions given that a spike is fired for sure within the interval \((t_i, t_{i-1}]\). The normalization imposed by such conditioning is necessary because of the neurons’ random firing as well as \( p \)'s explicit dependence on time, both of which generally prevent \( p(\Delta t, t_{i-1} + \Delta t) \) and \( q(\Delta t, t_{i-1} + \Delta t) \) from being properly normalized probability densities of \( \Delta t \), which in turn may render the KL divergence \( D \) a negative quantity. Plugging \( \text{D.2} \) into \( \text{D.4} \) and rearranging the terms, one obtains:

\[
C_{\text{batch}}(w) = \sum_{i=2}^{N} \int_{0}^{\infty} p^*_i(\Delta t) \cdot \left( \int_{0}^{\Delta t} h(t_{i-1} + \Delta t', w) d\Delta t' - \ln h(t_{i-1} + \Delta t, w) \right) d\Delta t
\]

\[
+ \sum_{i=2}^{N} \ln \left( Z_i(w) \right) - \sum_{i=2}^{N} H(p^*_i)
\]

\[
Z_i(w) := \int_{0}^{t_{i-1} - t_{i-1}} h(t_{i-1} + \Delta t, w) \cdot \exp \left( - \int_{0}^{\Delta t} h(t_{i-1} + \Delta t', w) d\Delta t' \right) d\Delta t
\]

where \( Z_i(w) \) is the normalization term of equation \( \text{D.6} \) and \( H(p^*_i) \) the differential Shannon entropy of \( p^*_i \).

In contrast to \( C(w) \), for \( C_{\text{batch}}(w) \) one can use time points between spikes as training examples by constructing a long time series \( p(\Delta t, t) \), which is reset at assumed random spike times \( T_s \), without actually simulating a spiking neuron with escape rate in online mode. This then allows \( C_{\text{batch}}(w) \) to be minimized in batch mode.
D.2 Details of the ESN Implementations

The following reservoir parameters have been used for the simulated Echo-State network of section 5.1.2 (see [67] for parameter definitions of the sigmoidal leaky-integrator neurons in the reservoirs):

**ISI Decoders** \((L_X, L_Y, R_X, R_Y)\)

Number of reservoir neurons = 300; reservoir sparsity (probability of connecting two reservoir neurons) = 0.10; spectral radius \(|\lambda_{max}| = 0.90\); leak \(a = 1\); time-step \(\delta = 1\); time constant \(c = 5\); scaling factor of the low-passed filtered spike inputs \(m_X \) and \(m_Y\): 0.80; scaling factor of the feedback connections from \(R_X, R_Y\): 0.80

**SPR Computer** \((L_{out}, R_{out})\)

Number of reservoir neurons = 1500; reservoir sparsity (probability of connecting two reservoir neurons) = 0.03; spectral radius \(|\lambda_{max}| = 0.97\); leak \(a = 1\); time-step \(\delta = 1\); time constant \(c = 2.5\); scaling factor of the inputs from \(R_X\) and \(R_Y\) to \(L_{out}\): 0.013; scaling factor of the low-passed filtered spike feedback from \(R_{out}\): 0.80

D.3 Details of the LSM Implementations

The basic setup of the LSMs simulated in section 5.2 is identical to that used for the population rate approach (see appendix section C.1). Different however were the dimensions of the reservoir (liquid-pool) and the neural synaptic parameters. In the following the values used for these quantities are provided, both for the gaussian 2-way factor and the sum-of-gaussians 3-way factor. Parameters not mentioned were set to the same values as in section C.1

Parameters of the LSM for the Gaussian 2-way Factor

Reservoir parameters:

Dimension of the liquid pool = \(6 \times 6 \times 15\); \(\lambda = 2\); \(C_{scale} = 1\); Dynamic synapses (with facilitation and depression) were used with the same parameter values as in [101]. Statistics of the synaptic parameters are listed in table D.1
Readout parameters:

\[ V_{\text{reset}} = 0 \text{ mV}; \quad V_r = 0 \text{ mV}; \quad V_\theta = 10 \text{ mV}; \quad C = 9 \text{ nF}; \quad R = 1\text{M}\Omega \Rightarrow \tau_m = 9\text{ms}; \quad T_{\text{ref}} = 3\text{ms}; \]

The standard deviation of the white noise process added to the drifting mean current was chosen such that a standard deviation \( \sigma_{V_m} = 8\text{mV} \) of the thus caused fluctuations of the membrane voltage \( V_m \) was induced. In contrast to the recurrent reservoir network we have used ordinary current-based synapses (without facilitation and depression) for connecting (a) the inputs to the reservoir and (b) the reservoir to the readouts. Parameter statistics of these connections are listed in table D.1. For the sake of simplicity the synaptic delays of the (learned) synapses from the reservoir to the readout population were set to zero, however, with an equivalent onset-shift of the rate pulses induced by the reference spikes during learning, such delays could have been trivially included.

<table>
<thead>
<tr>
<th>Connection Type</th>
<th>( \mu_W ) (nA)</th>
<th>( CV_W )</th>
<th>( \mu_D ) (ms)</th>
<th>( CV_D )</th>
<th>( \tau_s ) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>inside reservoir: EXC→EXC</td>
<td>60</td>
<td>0.7</td>
<td>1.5</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>inside reservoir: EXC→INH</td>
<td>120</td>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>inside reservoir: INH→EXC</td>
<td>-38</td>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>6</td>
</tr>
<tr>
<td>inside reservoir: INH→INH</td>
<td>-38</td>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>6</td>
</tr>
<tr>
<td>inputs to reservoir: EXC→EXC</td>
<td>60</td>
<td>0.7</td>
<td>1.5</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>inputs to reservoir: EXC→INH</td>
<td>120</td>
<td>0.7</td>
<td>0.8</td>
<td>0.1</td>
<td>3</td>
</tr>
<tr>
<td>reservoir to readout</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>70</td>
</tr>
</tbody>
</table>

Table D.1: Synaptic parameters of all connection types. \( \mu_W \) and \( \mu_D \) indicate the mean weight and synaptic delay respectively, \( CV_W \) and \( CV_D \) are the corresponding coefficients of variation. These parameters were used to assign random values to each synapse. \( \tau_s \) is the synaptic time constant, the '-' indicates learned synapses.

Parameters of the LSM for the Sum-Of-Gaussians 3-way Factor

The reservoir and readout parameters were mostly identical to those in the gaussian 2-way factor case. They differed only with respect to the connections from the message inputs to the reservoir, which were set to slightly smaller values (EXC→EXC: 40nA, EXC→INH: 80nA).

Miscellaneous Parameters and Setup of the Training Data

Simulation time step \( \Delta t = 0.2\text{ms} \); time constants of the pulse kernel (eq.5.2) added to the instantaneous firing rate after a reference spike (as used during the creation of suitable target time-series for learning):

\[ \tau_1 = 12\text{ms}; \quad \tau_2 = 10\text{ms}; \quad W = 50\text{ms}. \]

Training input data was created by piecewise homogeneous Poisson processes. That is, during each of 100 intervals of length 3s in simulated time, constant rates \( r \) were
determined such that a randomly and uniformly drawn \( P(1 \mid r, W) \sim U(0.1, 0.9) \) was matched according to eq. 3.31.
Bibliography


