On the Pattern Classification of Structured Data using the Neocortex-inspired Memory-prediction Framework

Anders Due Schwartz

Student:
Anders Due Schwartz (exam id: 173373)
Master of Science, Computer Engineering
Modern Artificial Intelligence

Master thesis project:
Period: February 1st 2009 - June 1st 2009
Counselor: Robert L. Jones III, PhD, Principal Scientist, Sigma Space Corporation
Counselor: John Immerkær, PhD, Associate Professor, University of Southern Denmark

Project performed at:
Sigma Space Corporation
4801 Forbes Boulevard
Lanham, Maryland 20706
USA

University:
University of Southern Denmark
Faculty of Engineering
Mærsk Mc-Kinney Møller Institute
Odense, Denmark
Abstract

In this master thesis project, we have researched how a theoretical model of the neo-cortex can be implemented as a hierarchical Bayesian network.

The report is based on the theoretical Memory-prediction Framework (MPF) by Hawkins & Blakeslee (2004), which was later implemented in the Hierarchical Temporal Memory (HTM) by George & Hawkins (2005). The assumption of the master thesis project is that the HTM is unable to implement fundamental concepts of the MPF and is furthermore based on methods and tools that do not scale well with complexity when they are applied to realistic and complex problems.

In this thesis we have been inspired by the work of Lee & Mumford (2003) and Dean (2006) in formulating an alternative model. The resulting novel Dynamic Hierarchical Nonparametric Belief Propagation (DHNBP) framework is based on the principals of the MPF framework and is able to facilitate representation of spatiotemporal sequences of features in a Dynamic Markov Network. The DHNBP framework is a novel extension of the Nonparametric Belief Propagation framework by Sudderth (2006) into hierarchies and time. In this report we provide algorithms for implementation, however, the DHNBP framework still has open-ended aspects that require further research.

Keywords: Dynamic Hierarchical Nonparametric Belief Propagation, time-varying Dirichlet process, Chinese Restaurant Process, Nonparametrics, particle filter, sequential Bayesian filtering, Gibbs and importance sampling, separation of causal and diagnostic support, slow feature analysis.
## Contents

1 **Introduction**  
1.1 Motivation ................................................. 1  
1.2 About the Project ......................................... 2  
1.3 Acknowledgement ........................................... 3  
1.4 Reading Guide & CD ......................................... 3  
1.5 Outline ..................................................... 3  

2 **Background**  
2.1 Memory-prediction Framework ................................. 5  
2.1.1 The MPF Model ........................................... 6  
2.2 Hierarchical Temporal Memory ............................... 10  
2.2.1 Reasoning under Uncertainty ............................ 10  
2.2.2 Bayesian Networks ...................................... 11  
2.2.3 Belief Propagation ...................................... 13  
2.3 Reasoning Strategies ....................................... 15  
2.4 Approaches to MPFs using Bayesian Networks .............. 16  
2.4.1 Hierarchical Temporal Memory (HTM) .................... 17  
2.4.2 Dean Hierarchical Hidden Markov Model (DHHMM) .......... 18  
2.4.3 Evaluation Summary .................................... 19  
2.5 Specification of the Alternative Solution .................. 19  
2.5.1 Inspiration ............................................... 19  
2.5.2 Reasoning using Particle Filtering ....................... 20  
2.5.3 Nonparametric Representation ........................... 20  
2.5.4 Network & Hierarchy .................................... 21  
2.5.5 Invariance .............................................. 21  
2.5.6 Application ............................................. 22  
2.5.7 Specification Overview and Goals ....................... 23  
2.6 Summary ..................................................... 24
3 Stochastic Simulation

3.1 Introduction .................................................. 25
3.2 Monte Carlo Integration ....................................... 27
3.3 Markov Chain .................................................. 27
3.4 Metropolis-Hastings Sampling Algorithm .................. 28
3.5 Single-Component Metropolis-Hastings ..................... 29
3.6 Gibbs Sampling ............................................... 30
  3.6.1 Factorizing the Joint Probability Distribution .......... 30
  3.6.2 Stationary Distribution ................................. 31
  3.6.3 Algorithm ............................................... 32
  3.6.4 Parallel Processing ...................................... 33
3.7 Importance Sampling ......................................... 33
3.8 Summary ....................................................... 35

4 Dirichlet Process

4.1 Introduction .................................................. 37
  4.1.1 Motivation ............................................... 38
  4.1.2 Nonparametric Distribution .............................. 38
4.2 Finite Mixture Model ......................................... 40
  4.2.1 Dirichlet Distribution .................................. 40
  4.2.2 Interpretation using the Pólya Urn ..................... 42
  4.2.3 Updating the model .................................... 42
4.3 Infinite Mixture Model ....................................... 43
4.4 Dirichlet Process ............................................. 44
  4.4.1 Dirichlet Process Mixture Model ......................... 45
4.5 Posterior and Predictive Dirichlet Process ................. 45
  4.5.1 Posterior Distribution ................................ 46
  4.5.2 Predictive Distribution & Blackwell-MacQueen Urn .... 47
4.6 Example: Chinese Restaurant Process Algorithm .......... 48
  4.6.1 Description ............................................ 48
  4.6.2 Terminology ............................................ 49
  4.6.3 Chinese Restaurant Process Algorithm ................ 50
  4.6.4 Alternative versions .................................. 51
  4.6.5 Prediction with the Chinese Restaurant Process ....... 51
4.7 Time-varying Dirichlet Process ............................. 53
4.8 Sub-conclusion .............................................. 54

5 Nonparametric Belief Propagation

5.1 Introduction .................................................. 55
5.2 Undirected Graphical model .................................. 57
## CONTENTS

5.3 Belief Propagation ................................................. 58
  5.3.1 Nonparametric Representation ................................. 59
  5.3.2 Message Propagation ........................................... 59
  5.3.3 Complexity of Message Product ............................... 60
  5.3.4 Approximating the Message Product ......................... 61
  5.3.5 NBP Message Construction Algorithm ....................... 62
5.4 Sub-conclusion ..................................................... 63

6 Hierarchical Nonparametric Belief Propagation 66
  6.1 Introduction .................................................. 66
  6.2 HNBP Features .................................................. 67
    6.2.1 Hierarchy .................................................. 67
    6.2.2 Input Nodes ................................................ 69
    6.2.3 Causal Feedback and the Dirichlet Process .................. 69
    6.2.4 Bayesian Analysis ....................................... 71
    6.2.5 Invariance in HNBP ....................................... 71
  6.3 HNBP Algorithm ................................................ 72
    6.3.1 Phase One: HNBP Message Construction .................... 72
    6.3.2 Phase Two: HNBP Belief Update ........................... 73
    6.3.3 Algorithm .................................................. 74
  6.4 Sub-conclusion ................................................ 74

7 Dynamic Hierarchical Nonparametric Belief Propagation 77
  7.1 Introduction .................................................. 77
  7.2 DHNBP Features ................................................ 78
    7.2.1 Time-varying Dirichlet Process ............................ 79
    7.2.2 Slow Feature Analysis ................................... 79
    7.2.3 Hidden Semi-Markov Model ................................. 81
    7.2.4 Invariance in DHNBP ..................................... 85
  7.3 Overview ..................................................... 85
  7.4 DHNBP Algorithm ................................................ 87
    7.4.1 Phase One: DHNBP Message Construction .................... 87
    7.4.2 Phase Two: DHNBP Belief Update ........................... 88
  7.5 Sub-conclusion ................................................ 89

8 Conclusion ........................................................ 91
  8.1 Conclusion ................................................... 91
  8.2 Future Research .............................................. 93
    8.2.1 DHNBP Algorithm ....................................... 93
    8.2.2 Learning .................................................. 94
    8.2.3 Application & Implementation ............................. 94
CONTENTS

A Approximation in Bayesian Networks 96
    A.1 Integrals in Bayesian inference ........................................ 96
    A.2 Gibbs Sampling ............................................................. 97
        A.2.1 Application .......................................................... 97
        A.2.2 Example ............................................................... 97
        A.2.3 Interpretation of the results ...................................... 101

B Research Plan 102
    B.1 Research Plan .............................................................. 102
    B.2 FORK Project .............................................................. 103
    B.3 Master Thesis Project .................................................... 105

C Summary of Schwartz (2008a) 107
    C.1 Introduction ............................................................... 107
    C.2 Conclusion ................................................................. 108

D Alternative Approaches 110
    D.1 Self-Organizing Maps .................................................... 110
        D.1.1 Introduction .......................................................... 110
        D.1.2 Adaptive Topology and the Discovery of Structural Form ...... 112
        D.1.3 ASSOM ................................................................. 114

E Miscellaneous 115
    E.1 Glossary ................................................................. 115
    E.2 Version ................................................................. 115
# List of Algorithms

1. Metropolis-Hastings Sampling Algorithm .................................. 29  
2. Single-Component Metropolis-Hastings Sampling Algorithm ............ 30  
3. Gibbs sampling with deterministic updating schedule ..................... 32  
4. Chinese Restaurant Process (Adapted from Ghahramani (2005)) ............ 50  
5. NBP Gibbs sampling of message product (simplified) ....................... 63  
6. NBP Message Construction .......................................................... 64  
7. Phase One: HNBP Message Construction of message $m_{k,j}$ ............. 75  
8. Phase Two: HNBP Belief Update of node $x_k$ .................................. 76
Chapter 1

Introduction

The focus of this master thesis project has been to investigate how probabilistic models of the human intelligence can be designed based on the work by Lee & Mumford (2003) and Hawkins & Blakeslee (2004). The proposed novel model is based on the concepts of the theoretical Memory-prediction Framework (MPF) by Hawkins and can be considered a theoretical alternative to the Hierarchical Temporal Memory (HTM) implementation of the MPF (George & Hawkins 2005). The HTM relies on so-called exact computation. However, this strategy was criticized in Schwartz (2008a). Thus, the objective has been to derive a model that incorporates the concepts of the MPF only based on more tractable simulation methods rather than using exact computation. The conclusion of this report is that the proposed model is successful in incorporating the relevant MPF concepts and is based on tractable simulation methods.

1.1 Motivation

The quest for intelligent computing is an ongoing effort. So far, to this author’s knowledge, it has not been possible to create truly intelligent systems as no methods have surfaced that are equal in performance to that of the human biological system for complex problems. Recently, much research focus has been directed towards models that are inspired by the biological features of the brain (in particular the neocortex) as a basis for building computational models that will run on computers (Hawkins & Blakeslee 2004).

Generally, computers have difficulties with interacting with the world. The main problem lies in their disability to deal with the variability and size of real world data, to analyze the data and make decisions based on that data. The data is simply too detailed and offers too many options for classification. However, it is clear that humans are able to deal with these difficulties and to make decisions based on highly noisy and ambiguous data. Such data can be classified often using many different plausible hypothesis. However, using their intellect, humans are often able to choose the correct explanation even when multiple causes seem plausible. The robustness and power of the human intellect is something that
no machine so far has been able to replicate. The idea by Hawkins & Blakeslee (2004) has been that if we are able to define and capture the fundamentals of the human intelligence, and then transfer it to computers, we could possibly achieve intelligent computing.

In Hawkins & Blakeslee (2004), Hawkins describes his theory on which relevant concepts of the human intelligence should be considered, when designing a computer model. The theory is named the Memory-prediction Framework (MPF) and is partly based on scientific facts and results, but also on assumptions made by Hawkins. In George & Hawkins (2005), Hawkins et al. describe how the theoretical MPF could be implemented as the Hierarchical Temporal Memory (HTM) model. The HTM model was evaluated in previous work by this author (Schwartz 2008a). The implementation was especially criticized for being constructed using “traditional” probabilistic methods, that do not scale well with complicated applications, but also for the model’s inherent lack of expressiveness (we will describe the problems facing the HTM in detail in Section 2.4).

In this master thesis project, we have been inspired by the work by Hawkins and generally accept his proposed MPF model. However, we recognize that the MPF is based on assumptions of how the neocortex functions, which is an area of much debate in the neuroscience community. It is however, the fundamental assumption of this thesis, that the concepts of the MPF are sound and relevant in achieving intelligent computing.

The goal of this master thesis has been to research how a theoretical model be developed that implements the concepts of the MPF, only using more expressive and tractable tools and methods. We will in Chapter 2 provide the reader with more in-depth background on the problems facing the HTM model. Then, in Section 2.5 we provide a detailed purpose description (goal list) of the project.

1.2 About the Project

This master thesis has been written for the University of Southern Denmark, Odense. The master thesis project has been a joint project between this author and Sigma Space Corporation of Lanham, Maryland, USA (suburb of Washington, D.C.). The goal has thus been to research how a model of the Memory-prediction Framework could be designed. Initially, a preliminary investigation (FORK project) into the master thesis problem domain was performed from September to December 2008. The findings of the FORK project were described in a separate report (Schwartz 2008a) and forms the theoretical foundation for this thesis. From February to June 2009 the master thesis project itself was carried out and was thus under a time constraint. Both the FORK and master thesis projects were carried out in the US at Sigma Space Corp. Because of the joint collaboration nature of the project, it has been necessary to define a common project that serves both Sigma Space Corp. and the student. In that relation, a research plan has been defined that describes the overall goals of the joint collaboration (see Appendix B).
1.3 Acknowledgement

As mentioned, the master thesis project has been researched in cooperation with Sigma Space Corporation. Principal Scientist Dr. Robert L. Jones III has provided a mentorship in where the student has found excellent guidance in the investigation of Memory-prediction Frameworks and related theory. Dr. Jones has offered his much valued guidance and experience in providing focus and inspiration to the ideas laid out in this thesis.

1.4 Reading Guide & CD

This report assumes the reader has an understanding of basic probability theory and more advanced topics as Markov chains, hidden Markov Models, particle filters, sequential Bayesian filtering and Bayesian belief networks. Chapter 2 provides the reader with technical background relevant for this report. However, as it is not possible to introduce all topics with equal detail, some referenced topics could prove unfamiliar to the reader. In such cases external references are generally provided by this author.

For the convenience of reader, the most important papers from the bibliography of this report has been provided on the accompanying CD in the directory “Papers”. The papers have been named after the template: “Author Year - Title.pdf”. Furthermore, the python implementation of the Gibbs sampler in Section A.2.2 is also available on the CD in the directory “Gibbs Sampler - Python Implementation”. Lastly, this report has been enclosed as a PDF in the directory “Report”.

A glossary in Appendix E.1 describes the most important terms used in this report.

1.5 Outline

The outline of this master thesis report is as follows:

In Chapter 2, we initially provide the reader with an introduction to the main concepts of the Memory-prediction Framework (MPF). Hereafter, we evaluate two implementations of the MPF and use the experiences gained in motivating our own solution. The chapter is concluded by stating the goals and objectives of the master thesis project.

From Chapters 3-7, we describe the solutions to the problems described in Chapter 2. We note that these chapters could prove to be technically challenging. Each chapter has been provided with a sub-conclusion (where applicable) that provides the reader with a conclusion and digest of the findings of the chapter. In Chapter 3 we describe the fundamentals of stochastic simulation with emphasis on the Gibbs and importance sampling algorithms. In Chapter 4, we describe the Dirichlet Process as a solution to implementing a nonparametric prior in a nonparametric setting. In Chapter 5, we describe the Nonparametric Belief Propagation (NBP) algorithm which is the foundation for the Hierarchical NBP (HNBP) in Chapter 6. The HNBP framework is a novel extension of NBP into a
hierarchy designed by this author. Then HNBP is then extended into time which yields the equally novel Dynamic HNBP (DHNBP) framework of Chapter 7.

The report is concluded in Chapter 8 with both an overall conclusion on the goals listed in Section 2.5 and a section with the author’s recommendations for future work.

Four appendices have been written to supplement the main report and can be read independently from the main report.
Chapter 2

Background

The purpose of this chapter is to provide the reader with the relevant background and basis for the main concepts of the theoretical Memory-prediction Framework (MPF) by Hawkins & Blakeslee (2004). Then to provide a specification list for an alternative solution to the Hierarchical Temporal Memory (HTM) implementation of the MPF by George & Hawkins (2005).

First, in Section 2.1 we describe the key concepts of the MPF model. We then extend our discussion towards the HTM implementation of the MPF in Section 2.2. Here we describe the key components of the HTM as being “traditional” probabilistic reasoning with focus on Bayesian networks and Belief Propagation. Hereafter in Section 2.4, a brief description and evaluation of two important approaches to implementing MPFs (the HTM and DHHMM models) are presented, based on earlier work by the author (Schwartz 2008a). The chapter then concludes with a specification list of goals for the master thesis in Section 2.5.

2.1 Memory-prediction Framework

The Memory-prediction Framework (MPF) is a unified theory by Hawkins & Blakeslee (2004) and is an attempt by Hawkins to devise a theoretical model that can be used to implement intelligent computing. In “On Intelligence” (Hawkins & Blakeslee 2004), Hawkins describes that in order to achieve intelligent computing, it is first necessary to define what human intelligence is. Then once the important concepts of human intelligence have been identified (no small matter itself), these concepts can be transferred to computers, thus resulting in intelligent computing. This is in fact what Hawkins tries to achieve in Hawkins & Blakeslee (2004). The book is both a description of how Hawkins understands human intelligence, but also a theory of how the human brain could work in order to facilitate the proposed human intelligence concepts. Hawkins draws on the work of neuroscientists, such as Mountcastle (1978), in justifying his claims of how, for instance, the physiological layout of the neocortex facilitates intelligent behavior. As research of the physiology of the
human brain is still an ongoing effort, the claims by Hawkins are partly based on results gained by researchers (such as Mountcastle), but also based on assumptions of how the human brain could work. In this master thesis we generally accept the claims made by Hawkins, mainly because the field of neuroscience research is outside our field of expertise and beyond the scope of this master thesis project. However, we realize that the MPF is a model based on assumptions and cannot be defined alone from the scientific research results currently available. That said, we believe that the ideas laid out by Hawkins are relevant and worth further research and investigation and are therefore the focus of this master thesis.

2.1.1 The MPF Model

The MPF is a model of how the human neocortex could work. Mountcastle (1978) proposes that the neocortex has an overall similar structure. Given that the neocortex handles input from many different sources such as smell, vision, etc., Hawkins & Blakeslee (2004) suggest that there could be one principal governing “algorithm” that handles the processing of all of the different input data of the neocortex. Thus, such an algorithm should be applicable to any kind of structured input data, independent of the origin of this data (i.e. sound, vision, tactile, smell and taste). When implementing such a model, one could divide the operations relevant to the MPF up into three parts as shown in fig. 2.1. The preprocessing stage of fig. 2.1 would provide the MPF with a uniform presentation of the raw input data invariant to what source provided the data. This allows for the design of a generic MPF algorithm that is not specific to any particular application. In this report we focus mainly on the design of the MPF algorithm itself and thus leave out the design of the pre- and post-processing stage.

The MPF implements four key concepts that are relevant to our discussion of the MPF model:

1. Hierarchy (part-based representation of objects)
2. Reasoning (learning, inference and prediction)
3. Invariant representations of objects

---

Figure 2.1: The MPF is encapsulated by a pre- and post-processing stage. The preprocessing stage ensures that the raw input data is manageable to the MPF, thus allowing the MPF algorithm to be generic and not specific to any particular application.
4. Spatiotemporal representations of objects

1. Hierarchy

A central part of the MPF is its ability to form representations of objects (structured data) by their parts. In fig. 2.2 an example of the part-based model can be seen. In the example, a bike has in Layer 1 been divided up into four major categories: tire, spokes, saddle and frame. These four objects can be combined to achieve the more complex and general object of bike in Layer 3. Thus, as we ascend the hierarchy of the parts (ascend the layers), the complexity of the model increases. We would normally assume that the MPF models objects that occur in some space (either physically or virtually). This assumption implies that an increase in complexity also means that parts at higher layers span over the accumulated space of the parts below them.

2. Reasoning

The second central part of the MPF is its ability to reason about the presence of each of the parts of the model at each layer. The reasoning model in the MPF is a probabilistic model, i.e. probabilities are used to represent the likelihood of any of the parts in the part-based model being present in the input data. For instance, let’s again assume the bike model of fig. 2.2. Now we further assume that new input data has become available, providing information about the presence (location) of all of the major categories of Layer 1. The MPF model can evaluate this data, i.e. the probability of the tire, spokes etc.
2.1 Memory-prediction Framework

Figure 2.3: Example of an invariant representation of a bike. The road, mountain and tandem bikes are interpreted (clustered) to be part of the same overall category (bike).

being present in the input data (known as inference), and pass each of the results upwards in the hierarchy to Layer 2. This information then serves as input for the parts on Layer 2. Again, we can evaluate these data and pass the results upwards to Layer 3, and so on. This process is known as bottom-up inference. Furthermore, information can also be passed down the hierarchy. By passing information down from higher layers of the hierarchy, parts in higher layers can help influence the probability of parts in lower layers being present. The process of passing information down the hierarchy is called top-down prediction.

Let’s assume we are currently only receiving conclusive input information regarding the tire, spokes and saddle from the input data, but input data regarding the frame is ambiguous (inconclusive). After a bottom-up inference pass, the bike part at Layer 3 is probable to be present in the input data (based on the fact that three out of four categories of Layer 1 are present). Similar to passing information bottom-up, the MPF model can now pass information down to the frame part in Layer 1, thus increasing the probability of the frame being present, even though there is no direct evidence to support the presence of the frame (other than the presence of the other bike parts). This way of using information from above to solve ambiguities below, is a fundamental concept of the MPF. Both inference, prediction and related concepts will be discussed further in Section 2.2.2.
3. Invariant Representation

The third central part of the MPF is its ability to form in\textit{variant} representations of objects. Again, we refer to the bike example of fig. 2.2. In this specific example, the model was constructed to represent a mountain bike based on mountain bike parts. However, say we instead want to be able to represent a generic bike that could represent any type of bike. Thus, when presenting the MPF with different types of bikes (e.g. mountain, road, tandem bikes etc.), they should all be grouped into the category \textit{bike} (as shown in fig. 2.3). This grouping is based on the fact that all bikes have certain parts in common (e.g. they all have tires, frames, saddles etc.). These parts can be identified in any instance of a bike. An MPF creates invariant representations of the bike parts by learning from different examples of bikes. If the MPF was presented to the parts (e.g. saddles, frames, etc.) from different bikes (e.g. mountain, road, tandem bikes etc.), it would form a single representation of each of the parts based on the different instances. For instance, a saddle representation would be a “mix” of saddles from mountain, road, tandem bikes etc. By forming invariant representations of objects, the MPF is able to recognize and group novel and familiar objects of similar type to be the same object.

4. Spatiotemporal Representation

The fourth central part of the MPF is an extension of the spatial invariant representation. Hawkins argues in Hawkins & Blakeslee (2004) that equally important to the ability to form stationary representations (of those described just above) is the ability to learn representations of objects that evolve over time. Thus, using time as a “teacher” to form spatiotemporal (space and time) invariant representations. Spatiotemporal representations are formed by associating parts (features) that occur contemporaneously. Furthermore, by learning sequences of parts, the MPF can remember sequences similar to what humans do when they recall a route they have driven, or a song they have learned. The extension of stationary inference and prediction into the time domain is important in order to create systems that are able to function in time dependent applications.

In reference to the bike example of fig. 2.2, we noted earlier that as the hierarchy is ascended, parts above accumulate the space of the parts below, thus increasing in complexity as the hierarchy is ascended. Therefore, as a consequence of extending the stationary spatial representation of the preceding section to a spatiotemporal representation, parts above also span across greater time spans compared to parts below. In order to account for this phenomenon, the MPF incorporates the concept of Slow Feature Analysis where parts in lower layers exist in finer time resolutions than parts in higher layers. This can also be interpreted as parts in lower layers change faster than parts in higher layers, so that parts in higher layers tend to be more stable in representation and exist for longer time periods than parts in lower layers. The time extension also has positive consequences for the reasoning ability of the MPF. As parts above span over greater time spans, they
2.2 Hierarchical Temporal Memory

In the preceding section, we described the fundamentals of the Memory-prediction Framework (MPF). The MPF is a theoretical model described by Hawkins & Blakeslee (2004) and is as such a theory and offers no implementation specifics.

In 2005 Hawkins co-founded the research company Numenta. Numenta’s mission is to design an implementation of the MPF. George & Hawkins (2005) describes an implementation of the MPF called the Hierarchical Temporal Memory (HTM).

The HTM is a machine learning model implemented as a so-called Bayesian network using the Belief Propagation algorithm by Pearl (1988). In this section, we proceed with a short technical introduction of the Bayesian network and the features of the Belief Propagation algorithm, as their concepts, properties and semantics are the foundation for this master thesis project. In Section 2.4 we evaluate two important implementations of the MPF (the HTM and DHHMM model). This evaluation serves as the motivation for this thesis as the purpose of this project is to improve on the HTM and DHHMM models and offer an alternative theoretical solution to the MPF. Such a solution would possess (at least) the same capabilities and functionality of the HTM and DHHMM, only using different tools and methods.

We start the description of the fundamentals of the HTM by first motivating the application of the Bayesian network by describing the ability to reason under uncertainty.

2.2.1 Reasoning under Uncertainty

It is the view of this author, that in order to construct powerful and flexible models, it is necessary to understand how humans reason. It is a part of people’s everyday life to make decisions based on incomplete and uncertain data. To be able to deal with incomplete data is a very important part of the human intellect as it enables us to make qualified guesses without having to understand the full scope of the problem at hand. Humans are able to abstract problems and quickly rule out factors that are not relevant to the current problem at hand and thereby simplifying matters. However, computer algorithms have traditionally had difficulties dealing with uncertainty. This has lead to algorithms that lack the flexibility of human reasoning, again leading to limitations in their application.

Pearl (1988) describes that there are generally two approaches to handling uncertainty: extensional systems (classical logic) and intensional systems (model based). The focus of this master thesis is intensional systems which can be implemented as a belief network. Belief networks provide the mechanism to identify what information is relevant for the
current problem at hand, and most importantly what is not, in a similar manner of how
humans quickly rule out the irrelevant. Belief networks consists of probabilistic nodes
(random variables) and their neighboring nodes are the relevant information to that node.
So by only referring to the neighborhood of the node we are able to rule out any other
nodes of the network as being relevant. This vastly simplifies computation.

Traditionally, in probability theory a belief network is known as a Bayesian network
or causal network. Bayesian networks are in essence inference engines for manipulating
relevance relationships, known in probabilistic terms as dependencies. The relevance
relationships are given as connections of conditional probability distributions between nodes
of the Bayesian network. With the relevance relationships encoded into the topology of
the Bayesian network, the system can ignore the irrelevant nodes by limiting the focus to
the connected nodes of the neighborhood. The concept of locality is very important and a
concept present in many related subjects to Bayesian networks and leads to more computa-
tionally tractable solutions. The combination of the model with relevance relationships
and probabilities in the Bayesian network provides a simple way of encoding knowledge by
the connections of the network and manipulating knowledge using bidirectional inference
as we describe next in Section 2.2.2.

2.2.2 Bayesian Networks

The Bayesian network is a type of probabilistic belief network that encodes causal
relationships between nodes and implemented as the framework of with reasoning is done in
the HTM implementation. A Bayesian network is a Directed Acyclic Graph\(^1\) (DAG) and
consists of random variables represented by nodes that are linked together with directed
edges (arcs). The graph of the Bayesian network expresses the joint probability distri-
bution of all the nodes of the graph. Often, Belief Propagation is applied to Bayesian
networks in order to perform bidirectional inference\(^2\). Belief Propagation is an algorithm
for reasoning under uncertainty, developed by Pearl (1988) and further explored by Neal
(1993). When used in conjunction with Bayesian networks the term “belief propagation”
is associated with the method developed by Pearl (1988) but is often, out of this context,
used more loosely to identify the distribution of knowledge in graphical models via some
method of bidirectional inference.

Bayesian networks are not the only application of belief propagation as implementa-
tions have also been developed for undirected graphs as Markov networks (Ott &
Stoop 2007, Sudderth, Ihler, Freeman & Willsky 2003, Isard 2003). Also known as causal
networks, Bayesian networks utilize the concept of hidden nodes that are causes of hier-
archically underlying nodes, so by grouping nodes hierarchically (child nodes have parent

---

\(^1\)Directed Acyclic Graph: Graphical model where there is no directed path that starts and ends on the
same node.

\(^2\)In this report we also refer to bidirectional inference as Bayesian analysis. Thus, the term Bayesian
analysis covers the application of Bayes’ theorem (see also eq. 4.1) to facilitate inference and prediction.
Bayesian networks is inference. Inference is the process inferring the state of a hidden node given that the children of the hidden node take on some values. Fig. 2.4 is an example of a simple Bayesian network where two nodes, $Z_1$ and $Z_2$, provide evidence (colored gray) for a hidden node $X$ (colored white). Inference is the application of Bayes’ theorem and has a graphical interpretation of reversing the arrows between the relevant nodes in the Bayesian network (see fig. 2.4). The resulting probability distribution for the hidden node based on the new evidence (input data) is also known as the posterior distribution.

A second core concept is prediction. Prediction is closely connected to inference and they can collectively be understood as bidirectional inference and implements a way of reasoning that is very natural to human beings. Prediction is inference “top-down” and the process of where nodes higher up in the hierarchy can pass information down to nodes below to influence and “sharpen” their beliefs, thus allowing nodes higher in the hierarchy to act as “teachers” for the nodes below. Furthermore, Bayesian networks are generative models, meaning they can reproduce (similar) previous input data based on the current beliefs of the network when no inputs are applied to the system. This means that the network can in essence recall or “play back” learned memories which bears a strong resemblance to the the concept of remembering stored sequences of experiences in humans.

A third core concept is “explaining away”. Explaining away is in essence the process of doing bidirectional inference. This concept is best illustrated using the example of fig. 2.5. In the example we have three (binary) nodes modeling a simplified system of factors influencing whether a car will start or not: Battery and Gas. Let’s assume that we are learning that the car won’t start. We are modeling two causes for that event: the battery is dead or the car has run out of gas. Let’s now assume that you remembered that you have just filled the car with gas. This means that Gas is clamped to true. This discovery suddenly makes it more likely that the battery being dead is the cause of why the car won’t
2.2 Hierarchical Temporal Memory

start. This concept is known as “explaining away”. The example of fig. 2.5 is very simple but illustrates that in order to “explain away” it is necessary to have so-called overlapping receptive fields, by letting nodes higher up in the hierarchy share the same nodes lower in the hierarchy. A biproduct of “explaining away” is the way that dependencies are induced in Bayesian networks. One could say that the state of the battery is independent of the amount of gas in the car but in the framework of Bayesian networks these two events are not completely independent of each other, but merely conditionally independent given the state of Car Start (Pearl 1988). When we confirm the value of one cause we indirectly also change the likelihood of the other cause which thereby “virtually” links the two causes together and is a natural way of reasoning that humans exercise on a daily basis.

The concept of “explaining away” leads us to the last important subject: belief propagation. There is a need for being able to propagate beliefs in the Bayesian network as evidence is available. When we learned that the gas tank was full in the car-example, we needed a mechanism for propagating this evidence, first down to Car start then up to Battery. Pearl (1988) defined Belief Propagation for this purpose.

2.2.3 Belief Propagation

Belief Propagation (Pearl 1988) is an algorithm for updating and propagating beliefs in a Bayesian belief network using Bayes’ theorem and implemented in the HTM as the method of reasoning. It is an algorithm with distributed local computation processors using the relevance relationships of the network to communicate with each other. The aim of the network is to achieve a belief equilibrium, given a change in belief is introduced somewhere in the network. The network then propagates messages around until a new equilibrium is achieved and the network settles with its new beliefs.

Belief Propagation is another mechanism that utilizes the concept of locality. Its distributed computation paradigm utilizes the independencies encoded into the graph (the connections of the Bayesian network), which simplifies the computation that leads to the
new equilibrium, substantially. Included in the paradigm is the ability to prevent positive feedback loops. As beliefs are passed between nodes it is very important that evidence is not passed to nodes from where it originated. If evidence is echoed back to the originating sources their beliefs would falsely be strengthened. However, built into Belief Propagation is a scheme to avoid counting the same evidence twice.

Originally designed for tree structures, Belief Propagation has also been expanded to polytrees and graphs with loops (see fig. 2.6). However, the HTM model has only been implemented to facilitate tree structures even though the Belief Propagation algorithm supports polytree structures (without loops). MPFs should ideally be able support arbitrary graph structures but often designers choose to limit the topology of their network to a specific type, say a tree or a polytree. But only dealing with tree structures or even polytrees in MPFs is a severe limitation to the expressiveness of the MPF. Being able to deal with loops is important because often graphs in realistic applications are loopy. However, reasoning in loopy graphs is potentially problematic given that beliefs could potentially never converge. We will return to this subject in the following section (Section 2.3).

Although Belief Propagation is an elegant solution to reasoning under uncertainty it faces problems when applied to real world problems and thus limits the strength of the HTM implementation. Performing exact Bayesian Belief Propagating is an NP-hard problem which means that computation time increases exponentially in the number of nodes. When trying to model real world problems, the graph size often grows to where computation using exact Bayesian Belief Propagation becomes intractable (Neal 1993, Andrieu, de Freitas, Doucet & Jordan 2003, Frey & Jojic 2005, Ihler, Fisher, Moses & Willsky 2004, Murphy 2001). The problem is that the joint probability distribution the graph encodes, can be complex and high-dimensional (see also Section A.1 of Appendix A). In practice, computing analytical solutions to these distributions is intractable. Thus, in this author’s opinion it is necessary to depart with the ways of “traditional” Belief
Propagation and research how solutions can be approximated instead of computed exactly.

2.3 Reasoning Strategies

In the following we continue our discussion of alternative reasoning schemes to the Belief Propagation algorithm which we started in the preceding section. As mentioned in the preceding section, “traditional” Belief Propagation suffers from two major shortcomings. Using Belief Propagation in large graphs becomes intractable, thus limiting the use of Belief Propagation. Furthermore, Belief Propagation is limited to graphs that are polytrees and thus not applicable to graphs with loops. However, loopy graph structures are potentially problematic. Normally, Belief Propagation in a tree or polytree would converge to some equilibrium, but when loops are present in the graph, beliefs can cycle (potentially indefinitely), thereby creating an unstable situation where the network never converges. Pearl (1988) lists three possible alternatives for reasoning in loopy graphs:

- Clustering
- Conditioning
- Stochastic Simulation

Furthermore, Pearl indicates a fourth solution (Pearl 1988, p. 195): simply use traditional Belief Propagation as is and hope the network will converge. This technique is known as Loopy Belief Propagation (Murphy, Weiss & Jordan 2000, Frey & Jojic 2005, Freeman & Pasztor 1999, Frey & MacKay 1998, Kschichang, Frey & Loeliger 2001). Determining if the network will converge is an empirical process and no analytical proofs of convergence exist. However, empirical tests show that often networks will in fact converge.

The first two techniques suggested by Pearl basically tries to break up the loops in the graph so traditional Belief Propagation can be applied. The latter stochastic method is the equivalence of Gibbs sampling (Lewicki & Sejnowski 1997) and provides a way of simulating the joint probability distribution of the graph. As mentioned earlier, when applying Belief Propagation to complex graphs, computation of beliefs becomes intractable. This is because Belief Propagation requires the exact computation of a normalizing constant. As an alternative to exact computation and reasoning in loopy graphs, stochastic simulation offers much promise (Pearl 1988, p. 211) for several reasons. Belief propagation using stochastic simulation:

1. is unaffected by the presence of loops in a graph, thus applicable to loopy belief propagation.

2. is approximate, thus providing approximations to true densities. The loss of precision gives way for computationally tractable belief propagation schemes that scales well with complexity.
3. is localized, meaning that only the neighborhood of a given node needs to be consulted when simulating (i.e. the Markov blanket).

4. is naturally a parallel message passing algorithm (as is traditional Bayesian Belief Propagation), thus lending itself to parallel processing implementation. However, the order of how the nodes in a graph is updated is not trivial. Pearl (1988) suggests a solution to the scheduling update problem (described in Section 3.6.4).

In summary, we note that belief propagation inside a Bayesian network can be performed in several ways. The method of choice for the HTM has been to use Pearl’s Belief Propagation algorithm for reasoning in a tree structured Bayesian network. However, alternative methods exist, such as belief propagation using stochastic simulation, which offers improved flexibility compared to exact computation using Pearl’s Belief Propagation algorithm. The most important improvement is the fact that stochastic simulation brings computational tractability to realistic and complex applications.

2.4 Approaches to MPFs using Bayesian Networks

So far in this chapter, we have described some of the main ingredients of the HTM implementation of the MPF model and discussed alternative reasoning schemes. In this section, we briefly evaluate two relevant neocortex-inspired implementations. This author has previously researched the following two models in Schwartz (2008a):3


2. Dean Hierarchical Hidden Markov Model (DHHMM) by Dean (2006) (DHHMM is based on the HHMM model originally by Fine, Singer & Tishby (1998))

Both models are based on research done by scientists in the field of neuroscience. For instance, both Hawkins and Dean refer to work by Hubel (1995) and Hubel & Wiesel (1962) on simple, complex and hypercomplex cells. These cells are present in the visual cortex of the human brain and provide a natural ability for humans to learn features present in input data from the receptive field, invariantly from transformations. Some of the major influences on Hawkins are the work done by Mountcastle (1978), Rao & Ballard (1999) and Creutzfeldt (1977) to mention a few. Major influences on Dean is Hawkins & Blakeslee (2004), Lee & Mumford (2003), Fukushima (1980), Földiák (1991), Ullman & Soloviev (1999) and Wiskott & Sejnowski (2002).

3This paper was written by the author for a course at the University of Southern Denmark as the findings of an investigation into two different approaches to MPFs. A summary of the paper can be found in Appendix C.
The author of this report has previously researched the HTM and DHHMM in Schwartz (2008a) in order to determine where the two models differ and what could be learned from them when designing an alternative solution. Although the two models differ in key areas, they are both Bayesian networks with a scheme for propagating beliefs in the network. The following two subsections provide descriptions of what was learned researching each of the approaches. For further descriptions on each of the models please refer to Hawkins & Blakeslee (2004), George & Hawkins (2005), Dean (2006) and Schwartz (2008a).

2.4.1 Hierarchical Temporal Memory (HTM)

The HTM approach by George & Hawkins (2005) is rooted in the book “On Intelligence” by Hawkins & Blakeslee (2004). In the book, Hawkins describes a unified and simplified theory of how the neocortex could work. Mountcastle (1978) proposed that the neocortex of the human brain is overall uniform in appearance and that the neocortex could be applying the same computational model to all parts of the brain (i.e. the visual, auditory, motor, etc. areas all perform the same basic operation). This way of abstracting the computational model of the neocortex is a novel way of constructing a computational model of intelligence that Hawkins uses in his HTM model.

In George & Hawkins (2005), a computational model of how the neocortex could be implemented, based on the concepts outlined in Hawkins & Blakeslee (2004), is described. In essence, George and Hawkins propose recognition of objects by using the Belief Propagation algorithm by Pearl (1988). The model was named “Hierarchical Temporal Memory” to indicate that

- nodes are arranged hierarchically (in a tree structure)
- time is an important part of learning
- that the nodes learn and remember input applied to the network

The conclusions in Schwartz (2008a) is that arranging nodes in a tree structure limits the expressiveness of the model. Specifically, it is not possible to implement “explaining away” in trees because nodes cannot have overlapping receptive fields (as this creates loops in the graph structure). However, by limiting the graph structure a number of computational advantages arise (e.g. Belief Propagation can be computed exactly when operating on trees which gives more precise results), but exact Belief Propagation becomes intractable for complex realistic problems. Previous implementations of the HTM model have not been able to perform prediction in the Bayesian network nor analyze the temporal dimension of data when performing inference. It is unclear if more recent versions (newer than 1.6) have resolved these issues but nevertheless, prediction and the ability to handle the time dimension is vital to an MPF. Both prediction and time has been described as important to obtaining intelligent systems in Hawkins & Blakeslee (2004).
2.4 Approaches to MPFs using Bayesian Networks

2.4.2 Dean Hierarchical Hidden Markov Model (DHHMM)

The DHHMM model by Dean (2006) is based on the same main principles as the HTM (in fact Dean refers to the work of Hawkins & Blakeslee (2004)) but without many of the restrictions of the HTM model. For instance, the DHHMM model is not limited to tree structures but supports polytrees. This gives the network support for “explaining away” and allow for overlapping receptive fields. Dean argues that overlapping receptive fields are important for (derived from Ullman & Soloviev (1999)):

1. Solving ambiguity
2. Creating invariant representations
3. Preventing models from over-fitting

The Bayesian network in the DHHMM model combines space and time in one composite spatiotemporal model. The network is implemented as a hidden Markov model which addresses the notion of time. Dean uses the concept of locality by introducing subnets. Each subnet has one hidden node that represents the state of all the nodes in the subnet. Using the receptive nodes in each subnet the hidden node abstracts the receptive nodes (providing spatial pooling). The hidden node is inserted into the hidden Markov model (thereby creating a spatiotemporal model, see fig. 2.7). Bidirectional inference is achieved using generalized Belief Propagation which when run on loopy Bayesian networks is approximate but reduces to exact under tree structured Bayesian networks.
2.5 Specification of the Alternative Solution

The DHHMM is a very attractive and elegant theoretical model, but when implemented it has so far only showed modest results (Dean 2006), suffering from the curse of dimensionality. This however, is a general problem with MPFs which also affects the HTM model. As the number of nodes grows, the need for processing power and memory increases (often exponentially) making the models computationally intractable when applied to real world problems.

2.4.3 Evaluation Summary

Common for both the HTM and DHHMM model is that they are both based on Bayesian networks. Although the DHHMM model implements belief propagation algorithms for both space and time (which the HTM has not been able to fully implement at the time of writing the thesis report), the implementation has unfortunately only showed modest results. It would seem that when both the HTM and DHHMM model are applied on realistic problems (i.e. large complicated data sets) they both have difficulties handling the increase in complexity. Thus, so far, they are mainly applicable to simplified “toy” problems.

In this author’s opinion, in order to create computationally tractable systems, it is necessary to depart from the idea of performing belief propagation exactly, and instead perform belief propagation using approximative techniques where solutions are approximations to the true solutions. Such techniques offer computationally tractable methods that scale well as complexity increases in applications. Thus, a trade-off exists between tractability and precision. In this master thesis project the idea is therefore to devise an alternative model implementing the same concepts as described in the MPF model but only using computationally tractable tools and methods.

2.5 Specification of the Alternative Solution

As mentioned in the preceding section (Section 2.4) the goal of this master thesis is to design an alternative to the HTM model as designed by George & Hawkins (2005). More specifically, the goal is to design a solution that is based on computationally tractable methods (i.e. using stochastic simulation), that still incorporates the concepts of the MPF as described in Section 2.1. In the next section we describe some of the major influences in the design of our own solution.

2.5.1 Inspiration

As mentioned earlier, this master thesis is founded on the work of Hawkins & Blakeslee (2004) and Dean (2006). We have found it relevant to seek inspiration in one of the original sources (Lee & Mumford 2003) that influenced both George & Hawkins (2005) and Dean
2.5 Specification of the Alternative Solution

(2005) in their solutions. Thus, in the remaining sections of this chapter we build our solution on ideas partially inspired by Lee & Mumford (2003).

2.5.2 Reasoning using Particle Filtering

Lee & Mumford (2003) describe their thoughts on how a computational model of the visual cortex could be based on hierarchical Bayesian inference. One of the main arguments made by Lee & Mumford (2003) is that a plausible brain model could be based on a hierarchical Bayesian framework and particle filtering mechanisms. A particle filter is a so-called sequential Monte Carlo method which performs iterative sequential Bayesian filtering using Markov Chain Monte Carlo techniques. Thus, by combining the hierarchical model of the MPF (and HTM) with particle filters for learning, inference and prediction in space and time (instead of using Pearl’s Belief Propagation), an alternative model can be specified. Such a model would be a departure from the exact computation of beliefs using Pearl’s Belief Propagation algorithm towards computing approximations to true solutions.

Particle filters were built for processing data that evolves in time, which is a vital part of the MPF that has not yet been fully implemented by the HTM implementation. As described by Hawkins & Blakeslee (2004), being able to learn and represent objects in space and time is paramount in achieving intelligent computing. Thus, implementing a reasoning scheme in space and time is of the utmost importance. In this author’s opinion, models such as particle filters would per definition fit the MPF model better than implementations of Pearl’s Belief Propagation algorithm in Bayesian networks. This is the case because particle filters, implemented in a causal network, are able to implement the concepts of the MPF from the beginning, in contrast to the HTM solution which was not from the beginning able to process time dependent data.

2.5.3 Nonparametric Representation

As a consequence of using particle filtering as a reasoning scheme in the MPF implementation, Lee & Mumford (2003) note that computation of posterior distributions using particle filters will be approximations to the true posterior probability distributions (in contrast to the exact computation of Pearl’s Belief Propagation algorithm in the HTM). Thus, such approximations will be computed as a discrete finite number of weighted samples (particles). In this report we build on on the idea of approximation and propose that all distributions are represented nonparametrically as a weighted set of mixture components of a parametric mixture model. Such a representation brings two major advantages:

1. The nonparametric model can be learned from the data. There is no need to specify the type of model a priori.

2. The model is only as complex as needed to fully explain the data, thus minimizing the risk of over- or under-fitting a solution to a model.
2.5.4 Network & Hierarchy

Using particle filters as a reasoning scheme implies defining a belief propagation scheme based on particles and nonparametrics. Lee & Mumford (2003) note that much potential lies in algorithms such as Nonparametric Belief Propagation (NBP) by Sudderth et al. (2003) and PAMPAS by Isard (2003). The NBP algorithm is ideally suited for belief propagation using the nonparametric approach.

As we have noted earlier, the MPF is a hierarchy of parts (or features). Thus, a belief propagation scheme will have to support inference from one layer of the hierarchy to another (both up and down, i.e. inference and prediction) indiscriminately. Lee & Mumford (2003, p. 1438) goes on to imply that algorithms such as the NBP and PAMPAS were designed to work in hierarchical domains. In this author’s opinion this is incorrect. In fact our research from the FORK project (Schwartz 2008b) showed that the NBP framework was developed for a flat one layered hierarchy of nodes and thus has no hierarchical support. However, given the positive features of NBP (described in Schwartz (2008b)), it was therefore intriguing to research if it was possible to extend the NBP algorithm to facilitate hierarchical support. This has therefore, been a goal of the master thesis project. We further describe the resulting novel Hierarchical Nonparametric Belief Propagation algorithm in Chapter 6.

Lee & Mumford (2003) goes on to note the importance separating feedforward and feedback streams in the implementation. Lee & Mumford (2003) hypothesize that as feedforward streams drive the generation of hypothesis bottom-up, feedback streams from higher layers can help provide prior distributions top-down for shaping inference at lower layers, thus allowing nodes at higher layers to be “teachers” for nodes at lower layers. As before, extending NBP to separate bottom-up and top-down streams would be a novel extension of the NBP algorithm. This has therefore also been an important part of the master thesis project and is also further described in Chapter 6.

2.5.5 Invariance

As mentioned earlier, the HTM model applies Pearl’s Belief Propagation algorithm as a reasoning scheme. In Section 2.4 we described that the use of Pearl’s algorithm implies limiting the network structure of the Bayesian network in the HTM to simple tree structures. However, by limiting the network to tree structures there is no support for so-called “overlapping receptive fields”. Overlapping receptive fields allow one node to have multiple parents (thus allowing parents to share the same data) but also give rise to loops in the network structure. As mentioned earlier, in order to facilitate “explaining away”, nodes need to be able to potentially have multiple parents. Thus, the network structure and belief propagation algorithm has to support loopy networks. Furthermore, as Dean (2006), we build on the work by Ullman & Soloviev (1999) that state that generalization and invariance is a product of analyzing multiple specific examples that overlap with the
novel stimuli\(^4\). Thus, by facilitating overlapping receptive fields and learning from multiple examples, we support the formation of invariant representations.

### 2.5.6 Application

The matter of applying the alternative solution to the MPF to application problems has not been a focus for this project. Thus, the nature of the project has alone been to investigate how a model of the MPF could be designed and not how it could be applied to application problems. Therefore, this thesis is purely theoretical and not practical. In this thesis we argue that before working on practical applications of models and algorithms it is first necessary to work out the theoretical foundation. However, early on in the master thesis project, some preliminary research was done in the area of how the MPF solution could be applied.

A popular implementation area of MPFs are computer vision problems. Both still images and video is a popular application of MPFs. Because MPFs are specialized in modeling features that evolve in space and time, video is an obvious application area that often otherwise can be difficult to analyze because of the space and time coupling. This area is currently being researched by Numenta, as they are developing applications for the HTM and computer vision.

As shown in fig. 2.1, when implementing the MPF, the system designer should place a pre- and post-processing stage before and after the MPF itself, respectively. This way, application specific details would be handled by the pre- and post-processing modules that ideally would offer the MPF implementation a clean interface so that a generic MPF module could be developed that would not be specific to any application. Thus, when applying the implementation of the MPF to a specific problem it would mainly be a task of designing suitable pre- and post-processing modules. The MPF can simply be considered a system that reasons about low-level features present in the input regardless of their source of origin. These features have to be provided by the preprocessing module. Thus, one could consider the preprocessing stage a feature extraction module. During research in this area, Scale Invariant Feature Transform (SIFT) descriptors has emerged as a popular and powerful tool for feature extraction (Lowe 2004). The SIFT descriptors provide scale and rotation invariant feature extraction. Thus combining the SIFT descriptors with the reasoning skills of the MPF seems as a good combination. Furthermore, in the FORK project, another relevant feature extraction algorithm was also discovered, known as Adaptive-Subspace Self-Organizing Map (ASSOM). The ASSOM is a special variant of the Self-Organizing Map and is able to learn transformation-, scale- and rotation-invariant filters (see Section D.1.3 of Appendix D for more details). Also the ASSOM seems promising to combine with the MPF. We conclude this short description of ideas for MPF application by noting that working on applying MPFs to real world

---

\(^4\)Sentence paraphrased from Ullman & Soloviev (1999).
problems is worthy of a master thesis study itself, and the focus of this report will solely be on the theoretical design of the MPF model.

2.5.7 Specification Overview and Goals

We now conclude our description of the alternative solution by providing a short overview of the most important concepts to implement, as described in Section 2.5.

Overall, the strategy of this project has been to depart with the ways of the HTM (which is a Bayesian network using Pearl’s Belief Propagation algorithm with limited temporal support). Instead, the overall goal has been to propose a hierarchical model for utilizing particle filtering techniques with nonparametric representation of distributions that natively support analyzing temporally dependent data. This overall goal can be divided into four categories (described in Section 2.5.2-2.5.5) that are summarized in the following list, together with subgoals of each category:

- **Belief Propagation**
  
  - The goal is to identify a method for performing belief propagation:
    * based on particle filtering techniques (Markov Chain Monte Carlo)
    * that is flexible enough to applied to arbitrary network topologies (incl. overlapping receptive fields)
    * across layers in a hierarchy but also time
    * as an extension of the Nonparametric Belief Propagation (NBP) algorithm
    * where NBP is novelly extended in the following areas:
      · To facilitate bidirectional inference in hierarchies and time
      · To help shape priors in layers below using recurrent feedback streams thus allowing parent nodes to “teach” child nodes

- **Nonparametric Representation**
  
  - All distributions are nonparametric approximations of their true densities
  - A suitable prior distribution has to be identified that supports nonparametric analysis that can be used in conjunction with NBP and feedback streams

- **Network & Hierarchy**
  
  - Objects should be represented by their parts (part-based model)
  - As the hierarchy is ascended, nodes accumulate space and time of nodes below

- **Invariance**
  
  - The model should have invariant spatiotemporal representation of objects
  - Invariant representations are based on overlapping receptive fields and learning from multiple examples
2.6 Summary

We conclude this chapter with a summary of the major findings.

In Section 2.1 we described the fundamentals of the theoretical Memory-prediction Framework (MPF). The MPF represents objects invariantly in a part-based hierarchical model. Furthermore, the MPF is able to represent objects over time, thus learning spatiotemporal (space and time) invariant representations. This amounts to learning sequences of objects.

In Section 2.2 we described the basics of the Hierarchical Temporal Memory (HTM) implementation of the MPF model. We noted that the HTM is implemented as a Bayesian network using Pearl’s Belief Propagation algorithm (Pearl 1988). The section ended with a short discussion of the problems of using Belief Propagation in complex settings.

In Section 2.3 we continued the discussion of Belief Propagation in Section 2.2 and described that belief propagation can also be performed using stochastic simulation as described by Pearl (1988), which is the foundation for this master thesis project.

In Section 2.4 we described the fundamental issues regarding both the HTM and the Dean Hierarchical Hidden Markov Model (DHHMM) (Dean 2005). The conclusion was that the HTM model is limited in its expressiveness mainly due to the fact that it is based on Pearl’s Belief Propagation algorithm.

In Section 2.5, we then described an alternative model to implementing an MPF based on the work by Lee & Mumford (2003). In the following chapters of this report we will describe solutions to the concepts of this alternative model.
Chapter 3

Stochastic Simulation

The purpose of this chapter is to introduce the Gibbs and importance sampling algorithms. Both sampling algorithms are very popular Markov Chain Monte Carlo (MCMC) algorithms and are integral parts of the Nonparametric Belief Propagation framework (introduced in Chapter 5).

The chapter initially describes the concept of Markov Chain Monte Carlo in its constituent parts (Monte Carlo Integration and Markov Chain). Hereafter, the derivation of the Gibbs sampler from the Metropolis-Hastings sampling algorithm is described. The chapter then concludes with a short introduction of the importance sampling algorithm.

For further introductionary and background sources on MCMC, refer to Bishop (2006), Gilks, Richardson & Spiegelhalter (1996), Tierney (1994), Andrieu et al. (2003). Note, readers that are familiar with the Gibbs and importance sampler can skip this chapter.

3.1 Introduction

As mentioned in Section 2.2.3, exact inference in Belief Propagation is often intractable when applied to real world problems. Therefore there is a need for an alternative method for computing probability distributions in belief networks. As an alternative, stochastic simulation methods offer computational tractability by approximating probability distributions.

In fig. 3.1, an overview of selected methods for performing inference in Bayesian networks can be seen. Approximate inference can be performed in many ways. Murphy (2001) lists three popular approaches as sampling methods, variational methods and Loopy Belief Propagation. The latter two are not the focus of this report.

Markov Chain Monte Carlo (MCMC) is a set of methods for approximating joint probability distributions using stochastic sampling methods. There are generally two classes of MCMC methods: Batch (offline) and sequential (online, also known as Particle Filtering). Both methods share a set of sampling algorithms (although slightly adapted versions for each class). A special case of the Metropolis-Hastings sampling algorithm, the Gibbs
sampling algorithm is a very popular MCMC method used in both batch and sequential MCMC and is the focus of Section 3.6. Gibbs sampling is especially suited for computing joint probability distributions in Bayesian networks using conditional probability distributions. The yellow path indicates the path of research in this project.

In batch MCMC (offline), all available data is used in the sampling process to compute the target distribution. If new data becomes available it is appended to the existing data set. When the sampling process is restarted both previous and new data is used to compute the new target distribution. The advantage is that the beliefs of the network are based on all available data leading to well based beliefs however computation time is longer (although not necessarily a problem). In contrast, sequential MCMC allows one to compute the online approximation of a probability distribution using samples (parti-
cles) of the distributions. Sequential MCMC methods are often applied to time sensitive applications where computation time is a constraint. By only working with approximate distributions the algorithm can bring down computation time.

3.2 Monte Carlo Integration

MCMC is the application of Monte Carlo integration using Markov chains. Monte Carlo integration is a stochastic method for evaluating integrals. As mentioned earlier in Section 2.2.3, these integrals occur in Bayesian networks (but also in many other contexts) and are often computationally intractable. In Bayesian networks the integrals occur when, for instance, marginalizing or computing the normalizing constant needed when applying Bayes’ theorem. However, as computationally intractable integrals are not limited to Bayesian networks we keep this introduction general (see Section A.1 for an introduction with Bayesian networks). Thus, in this section we introduce the general technique of evaluating integrals by sampling from them using Monte Carlo integration. In the following we formally describe the foundation for the Monte Carlo integration.

Let \( h(x) \) be a function that we wish to integrate over some interval \( a \) to \( b \):

\[
\int_{a}^{b} h(x)dx
\]  

(3.1)

\( h(x) \) can be decomposed into the product of some function \( f(x) \) and a probability density function \( \pi(x) \), so that:

\[
\int_{a}^{b} h(x)dx = \int_{a}^{b} f(x)\pi(x)dx = E[f(x)]
\]  

(3.2)

In eq. 3.2 we see that the integral of \( h(x) \) can be determined as computing the expectation of \( f(x) \) over the probability density \( \pi(x) \). In Monte Carlo integration the idea is to compute the expectation of \( f(x) \) in eq. 3.2 by drawing \( N \) (a large number) samples \( \{X^{(t)}\}_{t=1}^{N} \) from \( \pi(x) \) as:

\[
E[f(x)] \approx \frac{1}{N} \sum_{t=1}^{N} f(X^{(t)})
\]  

(3.3)

Thus, with Monte Carlo integration we are able to approximate the closed-form integration by drawing a large number of samples from \( \pi(x) \) and then computing their mean.

3.3 Markov Chain

The process of drawing the samples \( \{X^{(t)}\} \) used in eq. 3.3 remains to be determined. One way is to let the samples be drawn from a Markov chain that has \( \pi(x) \) as its stationary distribution. In that case this method of approximation becomes Markov Chain Monte Carlo.
Given that the samples \{X^{(t)}\} constitute a Markov chain they also hold the Markov property. This means that the next state \(X^{(t+1)}\) of the chain is only dependent on the current state \(X^{(t)}\) and not on the prior states \(\{X^{(0)}, X^{(1)}, \ldots, X^{(t-1)}\}\). The next state is sampled from the transition kernel \(P(X^{(t+1)}|X^{(t)})\) which defines the probability of going from \(X^{(t)}\) to \(X^{(t+1)}\).

With Markov chains we are required to specify a starting state of the chain, \(X^{(0)}\). However, the starting state of the chain has much impact on early samples drawn. As more samples are drawn, the transition kernel \(P(., X^{(0)})\) eventually stabilizes and reaches a steady state distribution \(\phi\) which approximates \(\pi\) (ideally \(\phi\) is exactly our sought after distribution \(\pi\)). \(\phi\) does not depend on neither the starting state \(X^{(0)}\) nor time \(t\). Loosely speaking, the chain gradually “forgets” its initial starting state and converges towards \(\phi\).

Many different strategies to determine the stationary distribution exists. Fig. 3.1 lists only a few of the most popular MCMC algorithms. The main focus of this chapter is Gibbs sampling, which is a special case of Metropolis-Hastings sampling.

### 3.4 Metropolis-Hastings Sampling Algorithm

The Metropolis-Hastings algorithm can be used to generate samples from a desired target distribution, \(\pi\). The stationary transition probability distribution \(\phi\), will converge towards \(\pi\) as more and more samples are drawn and inserted into the Markov chain. In order to construct the Markov chain, the Metropolis-Hastings algorithm uses a proposal distribution. If a sample from the proposal distribution meets an acceptance condition the sample is inserted into the Markov chain, otherwise it’s rejected.

Let state \(X^{(t)}\) denote the state of a Markov chain \(X\) at iteration \(t\). Then the next state \(X^{(t+1)}\) is determined by first sampling a candidate point \(Y\) from a proposal distribution \(q(\cdot)\). Candidate point \(Y\) is accepted as the next state of the Markov chain \((X^{(t+1)})\) with acceptance probability \(\alpha\):

\[
\alpha(X^{(t)}, Y) = \min \left( 1, \frac{\pi(Y)q(X^{(t)}|Y)}{\pi(X^{(t)})q(Y|X^{(t)})} \right)
\]

where \(\pi\) is the desired target distribution. The Metropolis-Hastings algorithm can be summed up as *Algorithm 1*. Note that \(X\) in *Algorithm 1* can be multidimensional. Generally, we define \(X\) as \(n\)-dimensional: \(\{X_1, X_2, \ldots, X_n\}\). This means that for each step of the chain all dimensions of the state are updated simultaneously in the Metropolis-Hastings algorithm (known as *en bloc* updating). A sometimes more computationally efficient algorithm is the *single-component Metropolis-Hastings algorithm*, where each dimension of the next state are updated one at a time.
3.5 Single-Component Metropolis-Hastings

With single-component Metropolis-Hastings we divide the updating of \( n \)-dimensional \( \mathbf{X} \) into \( n \) steps for each iteration of the algorithm. This means that only one dimension of \( \mathbf{X} \) is updated at a time but at the end of each iteration all dimensions of \( \mathbf{X} \) has been updated. This strategy gives way for a new proposal distribution. First, let \( \{X_{\backslash i}\} \) define the set of all dimensions of \( \mathbf{X} \) except \( X_i \):

\[
X_{\backslash i} = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n\} \tag{3.5}
\]

Each updating step \( i \), of iteration \( t \), composes of drawing candidate \( Y_i \) from \( q_i(Y_i|X_i^{(t)}, X_{\backslash i}^{(t)}) \). The acceptance probability becomes:

\[
\alpha(X_i^{(t)}, X_{\backslash i}^{(t)}, Y_i) = \min \left( \frac{\pi(Y_i|X_i^{(t)}, X_{\backslash i}^{(t)})q_i(X_i^{(t)}|Y_i, X_{\backslash i}^{(t)})}{\pi(X_i^{(t)}|X_{\backslash i}^{(t)})q_i(Y_i|X_i^{(t)}, X_{\backslash i}^{(t)})}, 1 \right) \tag{3.6}
\]

\( \pi(X_i^{(t)}|X_{\backslash i}^{(t)}) \) in eq. 3.6 is the full conditional distribution of \( X_i^{(t)} \) conditioned on \( X_{\backslash i}^{(t)} \). Note that, depending on what step the updating process is at, all components prior to component \( X_i^{(t)} \) have already been updated (if they were accepted using eq. 3.6), and all components after \( X_i^{(t)} \) still remain to be updated.

Algorithm 1 can easily be extended to the single-component Metropolis-Hastings algorithm as shown in Algorithm 2. The big advantage with the single-component Metropolis-Hastings sampler is that we can sample the joint probability distribution using only conditional probability distributions.

A special case of the single-component Metropolis-Hastings sampler is the Gibbs sampler. In Gibbs sampling we are exempt from having to compute the acceptance probability (eq. 3.6) which simplifies matters further.
Algorithm 2 Single-Component Metropolis-Hastings Sampling Algorithm

1: Initialize \( \{X_i^{(0)} : i = 1, \ldots, n\} \)
2: for \( t = 0 \) to \( T \) do
3: for \( i = 1 \) to \( n \) do
4: Sample \( Y_i \) from \( q_i(Y_i|X_i^{(t)}, X_{\backslash i}^{(t)}) \)
5: Sample \( U \) from uniform(0,1) r.v.
6: if \( U \leq \alpha(X_i^{(t)}, X_{\backslash i}^{(t)}, Y_i) \) then
7: \( X_i^{(t+1)} = Y_i \)
8: else
9: \( X_i^{(t+1)} = X_i^{(t)} \)
10: end if
11: end for
12: end for

3.6 Gibbs Sampling

As mentioned, the Gibbs sampler is a special case of the single-component Metropolis-Hastings sampler. The Gibbs sampler is essentially identical to single-component Metropolis-Hastings but the proposal distribution for the \( i \)’th component \( X_i^{(t)} \), at iteration \( t \) is:

\[
q_i(Y_i|X_i^{(t)}, X_{\backslash i}^{(t)}) = \pi(Y_i|X_{\backslash i}^{(t)})
\]  
(3.7)

When eq. 3.7 is inserted into eq. 3.6 the acceptance probability becomes 1 for all cases (Gilks et al. 1996, p. 12), thereby indicating that all samples drawn with the Gibbs sampler are inserted directly into the the Markov chain, regardless of their value. Furthermore, because the Gibbs sampler is based on the single-component Metropolis-Hastings sampler it is clear that generating samples using the Gibbs sampler only involves using conditional probability distributions.

3.6.1 Factorizing the Joint Probability Distribution

Consider the following joint probability distribution:

\[
p(X) = p(X_1, X_2, \ldots, X_n)
\]  
(3.8)

The variables \( X_i \) in \( \{X\}_{i=1}^{n} \) can be considered random variables in a graphical model, either an undirected (e.g. a Markov network) or directed graphical model (e.g. a Bayesian Belief Network).

When dealing with Bayesian belief networks, edges between vertices are directed, thereby encoding conditional relationships between vertices (random variables). The Gibbs sampler approximates the joint probability distribution by using the conditional
3.6 Gibbs Sampling

Figure 3.2: Example: Markov blanket for a node $X_1$ in a Bayesian network. The Markov blanket for a given node is defined as the parents, children, and co-parents of the node. In this case the Markov blanket consists of nodes $X_2$ to $X_7$. Nodes $X_8$ to $X_{17}$ are not part of the Markov blanket.

probability distributions. The joint probability distribution can be factorized as:

$$p(X) = \prod_{i=1}^{n} p(X_i | X_{\neg i})$$  \hspace{1cm} (3.9)

When wanting to compute the full conditional probability distribution $p(X_i | X_{\neg i})$ of a given node, it suffices (Pearl 1988, pp. 216-217) to only consider the Markov blanket\(^1\) of each of the nodes $X_i$ in $\{X_i\}_{i=1}^{n}$. This means that the full conditional probability distribution of a given node $X_i$ can be expressed using the Markov blanket as follows:

$$p(X_i | X_{\neg i}) = p(X_i | X_{pa(i)}) \prod_{k \in ch(i)} p(X_k | X_{pa(k)})$$  \hspace{1cm} (3.10)

where $pa(i)$ and $ch(i)$ denotes the parents and children of node $X_i$ respectively. Note that eq. 3.10 is only valid for a directed graphical model (for an example a Bayesian network). An example of a Markov blanket (indicated in yellow) for a node $X_1$ in a Bayesian network can be seen in fig. 3.2.

3.6.2 Stationary Distribution

As mentioned earlier, Gibbs sampling is an approximation of the joint probability distribution, $p(X)$, which is approximated by sampling from each of the full conditional probability

\(^1\)The Markov Blanket of a node $X_i$: $MB(X_i)$, is the parent, children and co-parents of $X$. The Markov blanket can also be seen as the set of nodes that renders $X_i$ independent of all other variables in the graph.
distributions. Each iteration of the sampling process can be interpreted as a state in a Markov chain (where each state is an \( n \)-dimensional vector for each of the sampled random variables in \( \{X_i\}_{i=1}^n \)). A transition distribution describes the probability of going from the current state at iteration \( t \) to any other state at iteration \( t+1 \), \( p(X^{(t)}|X^{(t+1)}) \). This means that we have to provide the Gibbs sampler with a starting state (initialization). However, the states of the chain are dependent on the starting state. Samples drawn from the target distribution from an early phase of the sampling process are highly dependent on the starting state and often can’t be considered as useable samples. Therefore, often a burn-in period is introduced wherein the samples are drawn from the target distribution but aren’t introduced as states in the Markov chain but simply rejected. After the burn-in period has completed, new samples drawn better represent the target distribution and can be used in the Markov chain. As more and more samples are generated, the chain will “forget” its initial state and \( p(\cdot|X^0) \) will eventually converge to a unique stationary distribution which is the desired target distribution and does not depend on starting state nor \( t \) (given that the modeled Markov chain is ergodic\(^2\)). So the longer the chain grows, the more accurate the approximation of the target distribution gets. This means that the approximation can be made as accurate as desired by increasing the number of samples drawn.

### 3.6.3 Algorithm

**Algorithm 3** Gibbs sampling with deterministic updating schedule

1: Initialize \( \{X_i^{(0)} \}: i = 1, \ldots, n \)
2: for \( t = 0 \) to \( T \) do
3: Sample \( X_1^{(t+1)} \sim p(X_1^{(t)}|X_2^{(t)}, \ldots, X_n^{(t)}) \)
4: Sample \( X_2^{(t+1)} \sim p(X_2^{(t)}|X_1^{(t+1)}, X_3^{(t)}, \ldots, X_n^{(t)}) \)
5: \(
6: \vdots
7: \)
8: Sample \( X_j^{(t+1)} \sim p(X_j^{(t)}|X_1^{(t+1)}, \ldots, X_{j-1}^{(t+1)}, X_{j+1}^{(t+1)}, \ldots, X_n^{(t)}) \)
9: \(
10: \vdots
11: \)
12: Sample \( X_n^{(t+1)} \sim p(X_n^{(t)}|X_2^{(t+1)}, X_3^{(t+1)}, \ldots, X_n^{(t+1)}) \)
13: end for

In **Algorithm 3** a deterministic version of the Gibbs sampling algorithm can be seen. As with a Markov chain, we provide the Gibbs sampler with a starting point (initialization) for each of the variables in \( \{X\}_{i=1}^n \). Hereafter we go through each of the relevant conditional probability distributions and sample from them. **Algorithm 3** can be considered deterministic in the way it cycles through the distributions. Here we simply sample each

\(^2\)A Markov chain is considered ergodic if any state is reachable from any state (irreducible) and that no cycles exist (aperiodic) in where the chain can cycle and get stuck.
of the distributions in the same order for each $t$. This will result in the chain having so-called global balance. Alternatively, one could use a random updating schedule, where a uniform random variable determines the order of update. This results in detailed balance. Gilks et al. (1996) even suggests that it is not necessary to update all of the components in each iteration; selecting each component with some probability. Zeger & Karim (1991) suggests to update highly correlated components more frequently than others to improve the quality of the samples drawn.

Algorithm 3 is simply an overall description of how the conditional probability distributions are used to compute the joint probability distribution of the network in question. However, there is no detailed description of how the sampling actually takes place. The example in Section A.2.1 in Appendix A addresses this concern.

3.6.4 Parallel Processing

Gibbs sampling is ideally suited for parallel implementation. It is possible to have several instances of Gibbs samplers working in parallel throughout the network in question. However, this do require some scheduling. It is important that each sampling process is based on fixed distributions. No neighboring nodes can be allowed to start a Gibbs sampling process at the same time. If they did, the resulting approximated probability distributions would not be correct. Pearl (1988, pp. 219-222) has a solution to this problem called edge reversal policy. This policy ensures that no neighboring nodes can start sampling processes simultaneously. The algorithm works by having a virtual version of the Bayesian network (not the real network) where arrows on all edges of the network are assigned arbitrarily. Each processor then inspects and wait until all arrows point towards the processor in question (making it a sink). Once all arrows point towards the processor, the sampling process can begin. When the processor has finished, it reverses all the arrows so they point away from the processor, thereby becoming a source.

3.7 Importance Sampling

Although the main focus of this chapter is the Gibbs sampler, we find it relevant to mention another “classical” and very popular sampling algorithm: the importance sampler. As in Section 3.2, let us again assume that the desire is to approximate the integral of eq. 3.2 by drawing a large number of samples ($N$) as in eq. 3.3, so that:

$$E[f(x)] = \int_a^b f(x)\pi(x)dx$$

$$\approx \frac{1}{N} \sum_{t=1}^{N} f(X^{(t)})$$

In order to evaluate eq. 3.12 it is necessary draw samples, $X^{(t)}$ from $\pi(x)$, however, this is not always feasible or possible. Instead we often have the case where although we cannot
sample directly from $\pi(x)$, we can however evaluate $\pi(x)$ for any $x$ up to an unknown normalization constant $Z_{\pi}$, so that:

$$
\pi(x) = \frac{\tilde{\pi}(x)}{Z_{\pi}},
$$

(3.13)

where, $\tilde{\pi}(x)$ can be easily evaluated. Given that we cannot work directly with $\pi(x)$ it is therefore more desirable to draw samples from a more well known proposal distribution, say $q(x)$, of our choice. The strategy of importance sampling is to generate samples from $q(x)$, rather than the one of interest $\pi(x)$, and then correct the bias (or error), introduced by sampling from $q(x)$, by using $\tilde{\pi}(x)$, which we easily can evaluate. We start by introducing $q(x)$ into eq. 3.11:

$$
E[f(x)] = \int_{a}^{b} f(x) \frac{\pi(x)}{q(x)} q(x) dx
$$

(3.14)

As mentioned above it is only possible to evaluate $\pi(x)$ up to a constant. This can often also be the case with the proposal distribution $q(x)$, so that $q(x) = \frac{\tilde{q}(x)}{Z_{q}}$. We change eq. 3.14 to reflect these facts:

$$
E[f(x)] = \frac{Z_{q}}{Z_{\pi}} \int_{a}^{b} f(x) \frac{\tilde{\pi}(x)}{\tilde{q}(x)} q(x) dx
= \frac{Z_{q}}{Z_{\pi}} \int_{a}^{b} f(x) w(x) q(x) dx,
$$

(3.15)

where $w(x) = \frac{\tilde{\pi}(x)}{\tilde{q}(x)}$, is known as the importance weight and corrects the bias introduced by sampling from the “wrong” distribution $q(x)$. We can now define the approximation to eq. 3.15 as:

$$
E[f(x)] = \frac{Z_{q}}{Z_{\pi}} \int_{a}^{b} f(x) w(x) q(x) dx
\approx \frac{1}{N} \sum_{t=1}^{N} w_{t} f(X^{(t)}),
$$

(3.16)

where $w_{t}$ is defined as:

$$
w_{t} = \frac{\tilde{\pi}(X^{t})/q(X^{t})}{\sum_{m} \tilde{\pi}(X^{m})/q(X^{m})}
$$

(3.17)

The normalization in eq. 3.17 ensures that the distribution of importance weights is in fact a proper probability distribution and is normally practical to evaluate.

Importance sampling algorithms are often easy to implement, however, this simplicity has a cost. The algorithms are sensitive to a number of parameters:
• The choice of proposal distribution $q(x)$ has a profound effect on the efficiency of the sampling procedure. If $q(x)$ is a good approximation to $\pi(x)$ then the error of sampling from $q(x)$ is small and vice versa. However, in realistic applications $\pi(x)$ is often unknown, so choosing the correct type of distribution for $q(x)$, a priori, can be difficult.

• If $f(x)\pi(x)$ in eq. 3.11 is complex and its mass is concentrated in small areas of $x$, it is likely that only a small number of weights will have a significant (or even non-zero) value. These few samples then dominate the entire distribution and is likely to lead to degeneracy problems in the sampling procedure. When $f(x)\pi(x)$ is highly complex, it is a real risk that the result from the sampling procedure is that no samples are drawn with non-zero weights. This leads to a total breakdown of the sampling procedure.

From the discussion above it is apparent that the importance sampling algorithm has inherent problems, however, its simplicity has much appeal and it therefore serves as a foundation for many other more robust algorithms. For instance, Detry, Pugeault & Piater (2009) applies the importance sampling algorithm as a two-level importance sampler, where the first level of importance sampling is used as a proposal distribution for the second level of importance sampling. In Isard (2003) the PAMPAS algorithm uses a simple importance sampling procedure combined with a resampling step to reduce the risk of degeneracy. Lastly, in Isard & Blake (1998) a more sophisticated temporal version of the importance sampler is implemented as the so-called condensation algorithm. The algorithm is a Sampling Importance Resampling (SIR) filter, when combined with the temporal analytical aspect, makes it a type of particle filter.

3.8 Summary

In this chapter we have described a number of popular tools used in stochastic simulation. In Section 3.2 we described Monte Carlo integration, which is the foundation for approximating integrals by drawing samples from them. Then in Section 3.3 we coupled Monte Carlo integration with Markov Chains and described Markov Chain Monte Carlo (MCMC). MCMC is a popular statistical method for approximating distributions by drawing samples from them (using Monte Carlo integration), where the sampling process is modeled as a Markov Chain.

The main focus of this chapter was the Gibbs Sampling algorithm (Section 3.6), which was derived from the Metropolis-Hastings Sampling algorithm. The Gibbs sampler is a very popular sampling algorithm which approximates a joint probability distribution by drawing samples the conditional probability distributions. This method is often preferable when dealing with situations where the joint probability distribution is too complex to compute exactly, but the parts of the joint probability distribution (the conditional
distributions) are readably accessible and tractable. Thus, the Gibbs sampler is often used to compute tractable approximations of otherwise computationally intractable joint probability distributions.

The chapter was concluded with a short description of the importance sampling algorithm. The importance sampler is also a very popular sampling algorithm that is able to weight samples drawn in the sampling process. Thus, some samples can be deemed more “important” than others. The importance concept is often utilized in particle filters where particles (weighted samples) approximate otherwise computationally intractable distributions. By weighting samples, arbitrarily complex distributions can be approximated. As we will see later, both the Gibbs and importance sampler is a vital part of the Nonparametric Belief Propagation (NBP) algorithm of Chapter 5 and our novel extensions hereof called Hierarchical NBP (HNBP) of Chapter 6 and Dynamic HNBP (DHNBP) of Chapter 7.
Chapter 4

Dirichlet Process

The purpose of this chapter is to give the reader an introduction to the Dirichlet Process (DP). The DP is a popular stochastic process often used as a prior distribution in iterative nonparametric Bayesian analysis. This chapter introduces the DP and related relevant theory in preparation for the application of the DP to the Hierarchical Nonparametric Belief Propagation (HNBP) framework in Chapter 6. We feel obligated to warn the reader that this chapter is detailed and possesses a reasonably high degree of difficulty. However, this is due to the inherent complex nature of the DP. This chapter should be seen as an attempt by this author to compile an introduction to DPs based on several sources (Jordan 2005, Ghahramani 2005, El-Arini 2008, Sudderth 2006). The chapter is divided up in four parts:

1. Introduction (Section 4.1)
2. Background theory (Section 4.2 and Section 4.3)
3. Core DP theory (Section 4.4, Section 4.5 and Section 4.7)
4. Example of DP sampling scheme (Section 4.6)

The last two parts are likely the most relevant parts of the chapter, and it is here the foundation and an example of application of the DP is laid out. For further information and background on the subject of DPs, please refer to Teh (2009), Gilks et al. (1996, pp. 459), Ranganathan (2006), Teh (2007), Neal (2000), Escobar & West (1995), Antoniak (1974).

4.1 Introduction

In this chapter, we introduce the Dirichlet Process. We start off by motivating its use.
4.1 Introduction

4.1.1 Motivation

As described in Section 2.5.2, an important task of a Memory-prediction Framework is perform Bayesian inference. In Bayesian inference we take some evidence $y$ into account to update or infer the belief (or posterior) $x|y$ that a given hypothesis $x$ will occur, based on some prior knowledge. In Bayesian analysis we compute the updated belief as Bayes’ theorem (eq. 4.1):

$$ P(x|y) = \frac{P(x)P(y|x)}{P(y)} , \quad (4.1) $$

where $P(x)$ is the prior belief (or knowledge) that was inferred before any new evidence was available. $P(y|x)$ is the likelihood distribution that returns the probability of seeing evidence $y$ given some specific hypothesis $x$. $P(y)$ is the marginal probability of seeing some evidence $y$ over the outcome of all possible hypothesis $x$.

Bayes’ theorem lends itself naturally to iterative updating schemes, such as particle filters (a focus of this report), due to the fact that the prior distribution $P(x)^n$ at iteration $n$ is updated to become the posterior distribution $P(x|y)^n$, which then again becomes the prior $P(x)^{n+1}$ for the next round of updating, as in eq. 4.2:

$$ P(x)^n \rightarrow P(x|y)^n \rightarrow P(x)^{n+1} \rightarrow \ldots \quad (4.2) $$

However, when constructing such an iterative updating scheme we have to specify the nature of the prior distribution, $P(x)$. As new evidence is taken in, the prior $P(x)$ and posterior $P(x|y)$ changes over updating iterations, as the prior augments new evidence with the previous knowledge and the result is a new arbitrarily complex distribution. Bayesian inference is most powerful if the posterior distribution $P(x|y)$ is able to accurately capture the nuances of the system we are trying to model through the Bayesian inference. The problem then remains, if we assume that the likelihood distribution $P(y|x)$ is well known, how do we choose a prior distribution $P(x)$ that ensures that the posterior distribution $P(x|y)$ accurately captures the properties of the system of interest? A simple (and in some cases naive) solution to this problem is simply to assume that the prior will always be of some well-defined parametric form (for instance we could assume that the prior would be a univariate Gaussian). However, in realistic applications we can often not a priori accurately specify the nature of the prior parametrically nor its parameters, and if we tried we would often incur intolerable errors and inaccuracies as it is hard to define a generic parametric model that fits all needs without the model over- or under-fitting.

4.1.2 Nonparametric Distribution

As mentioned earlier, it’s our assumption that in iterative Bayesian updating we realistically face arbitrarily complex distributions, so to use one specific predefined distribution
to model all instances would not suffice. Instead we are more interested in using a model where the parameters and the complexity of the model is learned from the data itself, and thus not defined in advance. This way the prior distribution can be updated to a richer posterior distribution by adding the needed complexity learned directly from the data in an unbounded way. The resulting posterior distribution would thus only be as complicated as needed to explain the data.

In this project the focus is to model continuous probability distributions as mixture models, by expressing a distribution as a convex combination of simpler parametric distributions. These parametric distributions are, in our case, simple one dimensional Gaussians with a mean $\mu$ and a variance $\Lambda$ (further described in Section 5.3.1). By creating nonparametric mixture model representations, we can express arbitrarily complicated non-linear and high-dimensional distributions by their parts (mixture components). The resulting convex combination distribution is represented as the weighted sum of its mixture components. Thus we avoid having to define the distribution by its closed-form expression (which in realistic applications is intractable). By adding and removing mixture components, we can define arbitrarily complex distributions in a tractable manner. Such a mixture model distribution does not in advance assume any specific type (for instance a 1D Gaussian distribution). In general, we denote models that don’t assume that the data is generated from any specific distribution as nonparametric models$^1$.

In summary, the desire is therefore to obtain a way of defining an appropriate prior distribution that iteratively can be updated (based on continuous input) in Bayesian analysis. The prior distribution should be modeled as a convex combination of mixture components (a mixture model) thus modeling arbitrarily complex distributions nonparametrically. As mentioned in Sudderth (2006), the Dirichlet Process (DP) incorporates the features we would like to facilitate. The DP is thus ideally suited for iterative Bayesian updating of nonparametric distributions and has therefore been chosen as a way of facilitating Bayesian inference and prediction in this project. Furthermore, the DP has an inherent clustering property that enables it to group or cluster continuous data into discrete clusters. This property is very useful in helping to provide invariant representations in the overall model as we shall see in Chapter 6.

We start our description of the DP by first describing the finite mixture model in Section 4.2. Hereafter, we expand the finite mixture model to infinite mixtures in Section 4.3. The infinite mixture model is the foundation for the Dirichlet Process which is formally described in Section 4.4. Lastly, we describe the popular sampling scheme, the “Chinese Restaurant Process” (an application of the Dirichlet Process) in Section 4.6.

$^1$Note that the term “nonparametric” does not, in this case, indicate that the models are free from parameters, but rather that they are flexible and not necessarily have to be fixed in advance and that the number of parameters grows with the data.
4.2 Finite Mixture Model

An inherent property of the DP, is its ability to cluster input data into categorial discrete clusters. Clustering is a very general problem in machine learning, artificial intelligence and statistics. A generative approach to clustering is simply to partition \( N \) data samples \( x = \{x_i\}_{i=1}^N \) in \( K \) clusters in a distribution \( \pi = \{\pi_1, \pi_2, \ldots, \pi_K\} \). Then, for each data cluster \( k \in \pi \), we create a cluster point. The cluster point then represents the distribution of members (data samples) of that cluster. This yields the finite mixture model as defined in eq. 4.3:

\[
p(x|\phi, \pi) = \sum_{k=1}^{K} \pi_k p(x|\phi_k),
\]

where \( \phi = \{\phi_1, \phi_2, \ldots, \phi_K\} \) is a set of mixture component parameters, and \( \pi = \{\pi_1, \pi_2, \ldots, \pi_K\} \) is the set of weight parameters for the mixture components. Thus, \( \phi \) and \( \pi \) are the parameters of the finite mixture model that we wish to learn, that in combination partitions all \( \{x_i\}_{i=1}^N \) data samples. We assume the same parameterized form for each mixture component \( p(x|\phi_k) \) (for the sake of simplicity) but with differing parameters. For instance, if \( p(x|\phi_k) \) is a Gaussian (for instance), with a mean and covariance so that \( \phi_k = (\mu_k, \Sigma_k) \), then the model would be a mixture model of Gaussians.

In mixture models, the assumption is that each data point belongs to only one mixture component. So therefore, the \( k' \)th data cluster is the distribution of data points arising from the \( k' \)th mixture component. We would ideally like to be able to model which data clusters (distributions) are responsible for generating which data points. This means that we need to be able to model a distribution over distributions (i.e. distribution over clusters). We can reformulate the distribution over clusters as distribution \( G \):

\[
G = \sum_{k=1}^{K} \pi_k \delta_{\phi_k},
\]

where \( \delta_{\phi_k} \) is an atom (Dirac delta cluster point) at \( \phi_k \). We now have a process of obtaining a sample from the finite mixture model. For \( N \) data samples we have the following model in fig. 4.1. As described in the caption of fig. 4.1, \( G \) is a discrete finite distribution of \( K \) clusters. However, one problem remains: How do we define \( \pi \) and \( \phi \)? The answer is to place prior distributions on these parameters as well in order to allow the parameters of \( G \) (and thereby \( G \) itself) to be random.

4.2.1 Dirichlet Distribution

As mentioned in the preceding section, it is necessary to further define the nature of the priors on \( \phi \) and \( \pi \). First, the prior on \( \phi \) is heavily dependent on application. In this report
4.2 Finite Mixture Model

Figure 4.1: Finite mixture model. Note, 1) $X \sim S$ means “$X$ has the distribution $S$” 2) We use “Plate” notation which is a “macro” that allow subgraphs (the contents of the square in fig. 4.4) to be duplicated (in this case $N$ times). The figure illustrates the process of obtaining a sample from the finite mixture model. $x_i$ is a data sample drawn from the distribution of $p(\cdot|\theta_i)$. Each $x_i$ is associated with a non-unique cluster $\theta_i$. In this example, we are modeling the relationship between $\theta_i$ and $x_i$ as a univariate Gaussian with $\phi_i = (\mu_i, \sigma^2_i)$, so that $p(\cdot|\theta_i) = N(\cdot; \mu_i, \sigma^2_i)$. Note that the full set of $\{\theta_i\}$ will likely contain duplicates, as some samples will be grouped to the same unique cluster $\phi_k$. Thus, there exists a subset of $\theta = \{\theta_i\}_N$ of size $K$ that maps each data sample in $x = \{x_i\}_N$ to a unique cluster in $\phi = \{\phi_i\}_K$. Furthermore, the non-unique cluster $\theta_k$ is drawn from the distribution $G$. Thus, $G$ can then be seen as a finite discrete distribution of unique clusters $\phi$. Leftmost in the figure, it can be seen that in this example, $G$ consists of three clusters. Each cluster is assigned to one or more $\theta_i$ variables which again is the cluster for some data sample $x_i$.

we limit the prior we place on $\phi$ to be a Gaussian, for the sake of simplicity\(^2\). We denote the prior on $\phi$ as $G_0$, the base distribution. Second, the prior distribution on $\pi$ should enable us to model a distribution of distributions (described in Section 4.2). For this purpose, the Dirichlet distribution is ideally suited. The Dirichlet distribution, denoted by $\text{Dir}(\cdot)$, defines a distribution of distributions. This means that a sample drawn from a Dirichlet distribution is a distribution itself. This is a departure from how we normally perceive probability distributions (i.e. samples drawn from distributions are usually scalars).

Formally, the Dirichlet distribution is a distribution over the $K$-dimensional probability simplex\(^3\). We denote $\pi = \{\pi_1, \pi_2, \ldots, \pi_K\}$ to be a $K$-dimensional point in the $(K-1)$-simplex. We denote $\pi = \{\pi_1, \pi_2, \ldots, \pi_K\}$ to be a $K$-dimensional point in the $(K-1)$-simplex. As described in Neal (2000), inference and prediction with the Dirichlet Process can be done by applying a simple Gibbs sampler to the DP distribution if $\phi$ is a conjugate prior to $p(x_i|\theta_i)$. If this is not the case, the process of inference and prediction becomes more complicated.

\(^2\)As described in Neal (2000), inference and prediction with the Dirichlet Process can be done by applying a simple Gibbs sampler to the DP distribution if $\phi$ is a conjugate prior to $p(x_i|\theta_i)$. If this is not the case, the process of inference and prediction becomes more complicated.

\(^3\)A $K$-dimensional simplex is the $K$-dimensional analogue of a triangle (a 2-simplex). More specifically, the simplex is the convex hull of a set of $K+1$ points. For example, a 1-simplex is a line segment, a 2-simplex is a triangle, etc.
simplex, so that $0 < \pi_k < 1$ and $\sum_{k=1}^K \pi_k = 1$. Further, we let $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_K\}$ be a set of parameters where $\alpha_k > 0$. We then define the Dirichlet probability density function as:

$$p(\pi|\alpha) = \frac{\Gamma(\sum_{k=1}^K \alpha_k)}{\prod_{k=1}^K \Gamma(\alpha_k)} \prod_{k=1}^K \pi_k^{\alpha_k - 1}$$

(4.5)

$$= \text{Dir}(\alpha)$$

(4.6)

where $\Gamma$ is the factorial Gamma function and $\pi_K = 1 - \sum_{k=1}^{K-1} \pi_k$. It is custom to denote the Dirichlet density with hyperparameters $\alpha$ as $\text{Dir}(\alpha)$.

Often, we don’t have any information about how each of the parameters $\alpha_k$ of $\alpha$ are distributed amongst the $K$ clusters, so we often use a symmetric Dirichlet prior where the hyperparameters are set symmetrically: $\alpha_k = \alpha_0/K$, where $\alpha_0 \triangleq \sum_{k=1}^K \alpha_k$.

As mentioned earlier, the Dirichlet distribution is especially suited as a prior in the finite mixture model as it is a distribution of distributions, where each distribution is a cluster. However, it can be difficult to intuitively understand the clustering properties of the Dirichlet distribution from eq. 4.5, so we provide the following interpretation example known as the Pólya Urn.

### 4.2.2 Interpretation using the Pólya Urn

The Pólya Urn is a thought experiment that can be used to illustrate the properties of the Dirichlet distribution. Consider an urn containing balls of $K$ different colors. The initial configuration of balls in the urn is $\alpha_1$ balls of color 1, $\alpha_2$ balls of color 2, $\ldots$, $\alpha_K$ balls of color $K$. Let’s now assume that we are going to draw $N$ balls from the urn. However, every time we draw a ball, we place the ball back and add an extra ball of the same color to the urn. This means that every draw (with replacement) also changes the distribution of the balls. As we draw more and more balls (and $N \to \infty$) the distribution of colored balls will be distributed as $\text{Dir}(\alpha_1, \alpha_2, \ldots, \alpha_K)$. The Pólya Urn experiment highlights an important property of the application of the Dirichlet distribution. As a ball of color $k$ is drawn, that ball is placed back and another one is added. This means that the chance of drawing a ball of color $k$ next has increased, resulting in a clustering effect where “popular” colors are chosen with higher probabilities (a “rich-get-richer” scheme).

### 4.2.3 Updating the model

We now expand our model of fig. 4.1 with the newly defined priors on $\phi$ and $\pi$ to fig. 4.2. In fig. 4.2 we now have added the two priors on $\pi$ and $\phi$. Note that $G$ is now distributed as a random discrete distribution with hyperparameters $G_0$ and $\alpha$. 
4.3 Infinite Mixture Model

So far we have described the finite mixture model. We have defined a model of a finite number of mixture components where each component is a distribution and can be interpreted as a cluster. However, defining an a priori suitable number of clusters can be difficult (i.e. defining $K$ in fig. 4.2). In the spirit of nonparametrics we would like the number of clusters to be potentially unbounded and then let the data determine the suitable number of mixture components. This means letting $K \to \infty$. We update eq. 4.4 to reflect this:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k} ,$$  \quad (4.7)

where

$$\phi_k \sim G_0$$ \quad (4.8)
$$\pi_k \sim \text{Dir}(\alpha_1, \alpha_2, \ldots, \alpha_K) , \quad K \to \infty \quad (4.9)$$

The resulting infinite mixture model can then be seen in fig. 4.3. Note that $G_0$ is a continuous distribution so the probability of that any two samples from $G_0$ are equal, is zero. However, $G$ is a discrete distribution with a countably infinite number of point masses (as indicated by the horizontal dots in fig. 4.3). Each of these point masses is in fact a distribution (or cluster).
4.4 Dirichlet Process

The infinite mixture model described in the preceding section (Section 4.3) is in fact the foundation for the Dirichlet Process (DP). We now proceed to expand the infinite mixture model of fig. 4.3 to the DP\textsuperscript{4}. First, we start by defining the state space on which the DP is defined. We let $G_0$ be a probability distribution on some measurable state space $\Theta$ (see example in fig. 4.4). Then consider a finite partition of the state space in $K$ sections of $\Theta$, so that:

\[ \bigcup_{k=1}^{K} T_k = \Theta, \quad T_k \cap T_j = \emptyset, \quad k \neq j \quad (4.10) \]

\textsuperscript{4}We generally follow the structure as in theorem 2.5.1. in Sudderth (2006, p. 96) and Jordan (2005).
The random probability distribution $G$ on $\Theta$ is then drawn from a DP if its distribution on each finite partition follows a Dirichlet distribution:

$$(G(T_1), \ldots, G(T_K)) \sim \text{Dir}(\alpha G_0(T_1), \ldots, \alpha G_0(T_K))$$

Then for any base measure $G_0$ and concentration parameter $\alpha$, there exists a unique stochastic process, called a Dirichlet Process (DP), satisfying these conditions, which we denote by $\text{DP}(\alpha, G_0)$. To denote that $G$ is distributed as a DP, it is custom to use the notation as in eq: 4.12:

$$G \sim \text{DP}(\alpha, G_0)$$

As mentioned in the preceding section (Section 4.3), samples $G$ from the Dirichlet Process (eq. 4.12) are discrete with probability one (Sethuraman 1994). As $G$ is drawn from a continuous distribution ($G_0$) we can draw a countably infinite number of distributions (or clusters) from $G_0$. This follows the nonparametric approach in that the number of clusters is theoretically unbounded and the data can determine how many clusters is appropriate, eliminating the need for determining the number of clusters a priori with the risk of over- or under-fitting.

As we will see in Section 4.6 we can exploit the discrete property of $G$ to cluster data because as samples $G$ are discrete with probability one, there is a non-zero probability of two samples colliding in the future. By adjusting $\alpha$ we can control the magnitude of this non-zero probability of collision. In fact, as $\alpha \rightarrow \infty$ then $G \rightarrow G_0$ (i.e. the larger $\alpha$, the more $G$ looks like $G_0$, the more clusters).

4.4.1 Dirichlet Process Mixture Model

When combining the infinite mixture model with the properties of the DP, we derive an application of the DP called the Dirichlet Process Mixture Model (DPM) (Antoniak 1974, Ferguson 1973). The DPM model of fig. 4.5 is then:

$$G \sim \text{DP}(\alpha, G_0)$$

$$\theta_i \sim G$$

$$x_i \sim p(x_i | \theta_i)$$

The DPM is thus suitable as a prior distribution for mixture models of a priori unknown complexity as there is no limit to the number of components that can be modeled (and generated). In the next section we expand our description of the Pólya Urn interpretation of the Dirichlet distribution of Section 4.2.2, towards the Dirichlet Process Mixture.

4.5 Posterior and Predictive Dirichlet Process

So far we have focused on the properties of the DP as a prior distribution. But as mentioned in the introduction to this chapter, the DP should also be able to take new evidence
Figure 4.5: Dirichlet Process Mixture Model (DPM). Combining the infinite mixture model with the properties of the Dirichlet Process we obtain an application of the DP (the DPM).

(a series of observations) into account to form a new posterior distribution based on the previous knowledge (from the prior distribution) and the new evidence. Furthermore, we are also interested in being able to predict future values of the DP by drawing samples from $G$. In the next section we describe the posterior and predictive DP.

### 4.5.1 Posterior Distribution

The DP distribution, partitioning the state space $\Theta$ in $K$ parts, was described in eq. 4.11 (repeated here for convenience):

$$ (G(T_1), \ldots, G(T_K)) \sim \text{Dir} (\alpha G_0(T_1), \ldots, \alpha G_0(T_K)) $$

(4.11)

We now wish to update $G$ with some new evidence $\theta_1, \ldots, \theta_N$, to obtain $G|\theta_1, \ldots, \theta_N$. When updating $G$ with new data we obtain a new DP because of the conjugacy\(^5\) between the Dirichlet and Multinomial distribution (Teh 2009):

$$ (G(T_1), \ldots, G(T_K))|\theta_1, \ldots, \theta_N, \alpha, G_0 \sim \text{Dir} (\alpha G_0(T_1) + n_1, \ldots, \alpha G_0(T_K) + n_K) $$

(4.16)

where $n_k$ is the number of data samples in the $k$’th cluster. This can also be written as (Sudderth 2006):

$$ G|\theta_1, \ldots, \theta_N, \alpha, G_0 \sim \text{DP} \left( \alpha + N, \frac{\alpha}{\alpha + N} G_0 + \frac{N}{\alpha + N} \sum_{i=1}^{N} \delta_{\theta_i} \right) $$

(4.17)

---

\(^5\)In general, a class of prior distributions (e.g. the Dirichlet distribution) is conjugate to another class of likelihood distributions (e.g. the Multinomial) if the posterior distribution is of the same class as the prior (e.g. Dirichlet).
thus, yielding the updated (posterior) $G$ distribution. From eq. 4.17 we can see the posterior $G$ is a result of both the previous $\alpha$ and previous $G_0$. We can further see that the new $\alpha = \alpha + N$. Thus every time the posterior is updated with $N$ samples, $\alpha$ increases with $N$.

4.5.2 Predictive Distribution & Blackwell-MacQueen Urn

The Blackwell-MacQueen Urn brings the foundation for the predictive DP (Blackwell & MacQueen 1973). The objective is to compute the probability of a future sample $\theta_{N+1}|G, \theta_1, \ldots, \theta_N \sim G$. This is done by computing the expected value of the posterior DP (eq. 4.17) with the posterior base distribution $G_0$ (Teh 2009):

$$P(\theta_{N+1} = \theta \in T|\theta_1, \ldots, \theta_N, \alpha, G_0 = E[G(T)|\theta_1, \ldots, \theta_N]$$ (4.18)

$$= \frac{1}{\alpha + N} \left( \alpha G_0(T) + \sum_{i=1}^{N} \delta_{\theta_i}(T) \right)$$ (4.19)

Eq. 4.19 follows the posterior base distribution $G_0$ of $G$ in eq. 4.17. Thus by integrating eq. 4.19 over $G$ we can marginalize out $G$ and obtain the posterior distribution of $\theta_{N+1}$:

$$\theta_{N+1}|\theta_1, \ldots, \theta_N, \alpha, G_0 \sim \frac{1}{\alpha + N} \left( \alpha G_0 + \sum_{i=1}^{N} \delta_{\theta_i} \right)$$ (4.20)

Notice that eq. 4.20 is similar to the posterior base distribution $G_0$ of eq. 4.17. Thus, the posterior base distribution is also the predictive distribution for the next sample $\theta_{N+1}$.

Drawing a sequence of predictive samples, using eq. 4.20 is known as the Blackwell-MacQueen Urn scheme. The basis of the Blackwell-MacQueen Urn is the ability to work on each of the category labels $\{\theta_i\}_K$ of fig. 4.5 separately. By marginalizing out $G$ we introduce dependencies between the $\theta$ variables (as shown in fig. 4.6). This means that when we draw a new sample $\theta_{N+1}$, this sample is conditioned on the previous samples.
θ_1, \ldots, θ_N$. Thus we can express the joint probability distribution of the samples as a product of their conditional distributions (as we did with the Gibbs sampler in Section 3.6.1):

\[
P(θ_1, θ_2, \ldots, θ_N) = \prod_{i=1}^{N} P(θ_i | θ_1, \ldots, θ_{i-1})
\] (4.21)

The random sequence of the θ samples are *infinitely exchangeable* (Teh 2009). This means that the joint probability distribution is invariant to any permutation of the order of the samples. Thus, given any permutation σ of 1, \ldots, N we obtain the same joint probability distribution:

\[
P(θ_1, θ_2, \ldots, θ_N) = P(θ_{σ(1)}, θ_{σ(2)}, \ldots, θ_{σ(N)})
\] (4.22)

As we saw in Section 3.6.1, permutable joint distributions of the type in eq. 4.21 and eq. 4.22 have joint probability distributions that can be simulated using MCMC techniques, such as the Gibbs sampler. By using MCMC techniques, it is possible to sample from the arbitrarily complex joint probability distribution using computationally tractable methods. This is in fact the foundation for the *Chinese Restaurant Process*, which we will discuss further in Section 4.6.

So far we have described the foundation for the DP, its inherent clustering property and its posterior and predictive distribution. However, the foundation itself is not an algorithm that readably can be implemented as computer code. The Blackwell-MacQueen Urn scheme is an extension of the Pólya Urn scheme described in Section 4.2.2 to the DP. In the next section we expand our discussion of the DP to one of the most popular DP sampling procedures, the Chinese Restaurant Process which is based on the Blackwell-MacQueen Urn scheme.

### 4.6 Example: Chinese Restaurant Process Algorithm

As mentioned in the preceding section, there is a need for inference and prediction algorithms based on the theory of the DP. As mentioned in Neal (2000), exact computation of posterior expectations (inference) for the DP is intractable for more than a few observations. Neal (2000) and Gilks et al. (1996) therefore suggests using a Markov Chain Monte Carlo (MCMC) sampling scheme for simulating the posterior and predictive density instead of computing it exactly. Ghahramani (2005) lists 5 of the most popular sampling schemes. The most intuitive and relevant DP sampling scheme (in this author’s opinion) is the *Chinese Restaurant Process*, which is the focus of this section.

#### 4.6.1 Description

The *Chinese Restaurant Process* (CRP) is a stochastic process (a Dirichlet Process) that explicitly shows the clustering property of the DP. The CRP is based on the infinite
exchangeability of the observations of the DP (as described in Section 4.5.2 and eq. 4.22) and the Blackwell-MacQueen Urn scheme of eq. 4.20. The CRP can both be used for posterior updating and predictive simulation using eq. 4.20 as a foundation. Since the value of draws from eq. 4.20 are repeated, we now change eq. 4.20 to represent only unique clusters in preparation for the CRP algorithm. We denote the set of unique clusters as \( \{\theta^*_k\}_K \), which is a subset of the non-unique set of clusters \( \{\theta_i\}_n \), so that \( K \leq n \), and we denote \( n_k \) the number of repeats of \( \theta^*_k \) (i.e. the number of data samples belonging to cluster \( k \)). The predictive distribution now becomes eq. 4.23:

\[
\theta_{N+1}|\theta_1, \ldots, \theta_N, \alpha, G_0, \sim \frac{1}{\alpha + N} \left( \alpha G_0 + \sum_{k=1}^{K} n_k \delta_{\theta_k} \right)
\]  

(4.23)

From eq. 4.23 we notice that when sampling a predictive value, \( \theta^*_k \) will be sampled with probability proportional to \( n_k \) (i.e. the number of times the cluster already has been observed/sampled). This means that the more we sample a cluster, the more likely we are to sample it again in the future (i.e. a rich-get-richer scheme). This is also the clustering property of the DP in effect. In fact, Teh (2009) notes that the number of clusters only grows logarithmically in the number of observations (\( K \rightarrow \alpha \log(N) \) as \( N \rightarrow \infty \)). This further stresses the clustering property of DP, but also shows the nonparametric property of the DP, in that it favors models whose complexity grows with the dataset size (Sudderth 2006). Furthermore, we also notice that the probability of drawing a new cluster is proportional to \( \alpha \). As \( \alpha \) is a parameter under the designers control, it is possible to manually bias whether new observations preferably should be assigned to new clusters (\( \propto \alpha \)) by using a large \( \alpha \) or be assigned to one of the \( K \) already existing clusters (\( \propto n_k \)) where \( n_k \) is determined empirically by the process itself (by simple addition). Generally, using a large \( \alpha \) results in the probability mass of \( G \) being spread out, leading to observations being clustered in many clusters. Vice versa, a small \( \alpha \) means that the probability mass is concentrated over fewer areas, thus resulting in observations being clustered in fewer clusters. We noted in context of eq. 4.17, that as the DP is updated with \( N \) observations \( \alpha \) increases with \( N \). This means that as more and more observations are included in the DP, \( \alpha \) increases and so does the number of clusters.

### 4.6.2 Terminology

The name Chinese Restaurant Process (CRP) was coined by Pitman and Dubin (Pitman 2002) and was inspired by the seemingly infinite seating capacity of Chinese restaurants in San Francisco (Sudderth 2006), thus referring to the unbounded nonparametric clustering property of the DP. In the CRP analogy, the restaurant is the DP itself. Customers entering the restaurant are analogous to observations and the tables seating the customers are the clusters/distributions/mixture components. In theory, the restaurant has infinite seating capacity (both infinite amount of tables and seats) but in practice seats a finite
number of customers at a finite number of tables, but has a capacity of a potentially unbounded number of guests.

4.6.3 Chinese Restaurant Process Algorithm

The CRP can be expressed as in Algorithm 4. Note, “w.p.” stands for “with probability”.

From Algorithm 4, we notice that each time a new customer enters the restaurant the total sample space can be defined as the sum of either selecting one of the existing $K$ tables ($\propto n_k$) or selecting a new table ($\propto \alpha$):

$$P(\theta_{N+1}|\theta_1, \ldots, \theta_N, \alpha) = \sum_{k=1}^{K} \frac{n_k}{\alpha + N} + \frac{\alpha}{\alpha + N}$$

(4.24)

$$= 1$$

(4.25)

Algorithm 4 Chinese Restaurant Process (Adapted from Ghahramani (2005))

1: # 1. Definitions:
2: Restaurant has countably infinitely many tables $k \in N_+$
3: Customers are indexed by $i \in N_+$, and $\eta_i$ is the table customer $i$ is seated at
4: Tables have values $\theta_k$ drawn from $G_0$
5: $K =$ the total number of occupied tables so far
6: $N =$ the total number of customers so far
7: $n_k =$ number of seated customers at table $k$
8: # 2. The CRP algorithm:
9: Initially, the restaurant is empty of customers. Now $N_{total}$ customers arrive.
10: Customer 1 enters the restaurant and sits at table 1
11: Thus: $\eta_1 \sim G_0$, $K = 1$, $N = 1$, $n_1 = 1$
12: for $n = 2, \ldots, N_{total}$ do
13:  customer $n$ sits at table $\begin{cases} k \text{ w.p. } \frac{n_k}{\alpha+N} & \text{k = 1, \ldots, K} \\ K+1 \text{ w.p. } \frac{\alpha}{\alpha+N} & \text{New table} \end{cases}$ (Existing table)
14:  if customer $n$ sits at a new table ($K+1$) then
15:  $K \leftarrow K + 1$
16:  $\theta_K \sim G_0$ (sample new table value from $G_0$)
17:  end if
18: $\eta_n = \theta_k$ ($k$ indexes the table customer $n$ sat at)
19: $n_k \leftarrow n_k + 1$
20: $N \leftarrow N + 1$
21: end for
4.6.4 Alternative versions

When reading the literature (Sudderth 2006, Ghahramani 2005, Jordan 2005, Teh 2007, Teh 2009, Gilks et al. 1996), it quickly becomes clear that although most authors agree on the fundamentals of the DP, many have slight variations in their algorithms or implementations. In Algorithm 4, new arriving customers are seated based on how many that are already seated at a table (a “social” factor) and the $\alpha$ parameter. However, as described by Gilks et al. (1996) it is also possible to instead seat customers based on how well they fit the value ($\theta_k$) of the table (i.e. a “compatibility” evaluation instead of a social evaluation). In this setting, new customers arriving would keep their original value and then simply be “linked” to the table. This way, it is always possible to evaluate how well the current set of customers fit the current table value (in Section 4.7 we will see that table values can be recomputed to better reflect the values of the currently seated customers).

In this report, we suggest the combination of the both the social and compatibility factor when evaluating the discrete sample space of all possible outcomes in Step 13 of Algorithm 4. Gilks et al. (1996) suggests using a Gaussian as a penalizing function that favors seating customers at a table with values that are similar to the customer itself. When adopting the two factor strategy, Step 13 of Algorithm 4 becomes:

\[
customer \, n \, \text{sits at table} \begin{cases} 
  k \text{ w.p. } \frac{n_k}{\alpha + N} e^{-\frac{(\eta_n - \theta_k)^2}{2\sigma_k^2}} & k = 1, \ldots, K \\
  K + 1 \text{ w.p. } \frac{\alpha}{\alpha + N} 
\end{cases} \\
(4.26)
\]

where, $\sigma_k^2$ is the variance associated with the Gaussian penalizing function for the $k$’th table.\(^6\)

We conclude our description of the CRP with an illustrative example in fig. 4.7. In this example we illustrate the process of determining the discrete posterior distribution of the CRP (i.e. the DP) when taking new observations into account (based only on the social factor for simplicity). For further details, please refer to the caption of fig. 4.7.

4.6.5 Prediction with the Chinese Restaurant Process

As mentioned in Section 4.5.2 the process of determining the posterior and predictive distribution is based on the same equation. Thus, the process of predicting a future value from the CRP is identical to the CRP algorithm described earlier, but with the exclusion of actually changing the distribution by inserting into it. The process of determining the predictive distribution of the CRP, given previous observations $\theta_1, \ldots, \theta_N$, can be done by drawing $T$ samples from $G$ as in eq. 4.27 (Gilks et al. 1996, p. 461):

\[
\theta^t_{N+1}|\theta_1, \ldots, \theta_N, \alpha, G_0, \sim \frac{1}{\alpha + N} \left( \alpha G_0 + \sum_{k=1}^{K} n_k \delta_{\theta_k} \right), \quad \forall t \in T \\
(4.27)
\]

\(^6\)Note, that eq. 4.26 is a simplified version of the one given by Gilks et al. (1996, p. 460).
4.6 Example: Chinese Restaurant Process Algorithm

Figure 4.7: Illustration of an example of a Chinese Restaurant Process. Circles represent tables (clusters). The fraction inside each table is the probability of a customer sitting down at the specific table. Diamonds represent customers. The number inside each diamond corresponds to the customer’s value. The table furthest on the right in each row (with no customers) symbolizes a new (undrawn) table to be drawn from an unbounded number of tables ($\theta \sim G_0$). Rows of step 1, 3 and 5 can each be interpreted as the DP distribution (or $G$) of customers and tables at the given step. **Step 1)** The current state of the CRP is distributed as two tables from earlier posterior updates. Table 1 has three customers and Table 2 has one. The probability of a new customer sitting down at Table 1 is $\frac{3}{\alpha+4}$, Table 2 is $\frac{1}{\alpha+4}$ and sitting down at a new table is $\frac{\alpha}{\alpha+4}$.

**Step 2)** A new customer enters the restaurant (with value 7 indicated by red diamond). **Step 3)** By sampling a table from the discrete distribution of Step 1), the new customer with value 7 is seated at Table 2 (this shows the stochastic nature of the DP as customers are seated with a probability, not a certainty). After seating customer with value 7 at Table 2, the discrete distribution changes to Table 1: $\frac{3}{\alpha+5}$, Table 2: $\frac{2}{\alpha+5}$ and a new table: $\frac{\alpha}{\alpha+5}$.

**Step 4)** A new customer enters with value 9. **Step 5)** The new customer is seated at a new table after sampling which table to sit at from the discrete distribution of Step 3). The new table is given a table value by sampling from $G_0$. The discrete table distribution has now changed to Table 1: $\frac{3}{\alpha+6}$, Table 2: $\frac{2}{\alpha+6}$, Table 3: $\frac{1}{\alpha+6}$ and a new table: $\frac{\alpha}{\alpha+6}$. Figure inspired by Sudderth (2006).
Hereafter, the mean of those $T$ samples represents the predicted value:

$$
\theta_{N+1} = \frac{1}{T} \sum_{t=1}^{T} \theta'_{N+1}
$$

Note, that computing the predictive density in this manner is approximative in practice, but as $T$ is increased, the quality of the estimate also increases. Furthermore, note that the combination of eq. 4.27 and eq. 4.28 is a Monte Carlo integration which yields an approximation of the true predictive density.

## 4.7 Time-varying Dirichlet Process

The DP described so far in this chapter is only applicable to modeling distributions that are stationary in time. When modeling distributions that evolve over time it is necessary to introduce extra steps to ensure the integrity of the DP is preserved at all times. In Chapter 7 we extend our algorithms towards processing time dependent data. We therefore conclude this chapter by extending the DP towards time series processing in preparation for Chapter 7.

Different strategies exist for expanding DPs into the time domain (Griffin & Steel 2006, Srebro & Roweis 2005). However in this report, we have adopted the strategy of Caron, Davey & Doucet (2007). As mentioned in Caron et al. (2007), a main problem with analyzing time dependent data using the DP, is that the stationary DP assumes an “accumulated” distribution of tables and customers:

1. When a customer has entered the restaurant the customer stays forever and never switches table
2. Once a table has been created it never disappears from the distribution again

Both concerns are unfortunate properties when working with continuous data that evolves over time. Often, in time dependent data, the case is that as time progresses, “old” customers (data samples) and tables (clusters) should have less influence on the overall distribution as the current input data to the DP no longer holds evidence of the “old” customers and their associated tables. Caron et al. (2007) suggests two strategies for preserving the randomness of the DP at any time in the process. When going from time step $t - 1$ to $t$:

1. Delete customers (data samples) randomly. If, after deletion, a table is empty, the table itself is deleted.
2. Randomly move surviving tables to new positions according to some distribution.

After the two steps of deleting customers and moving tables have been executed, the randomness of the DP has been preserved and the process of adding new customers to
the DP (as in the CRP in Step 13 of Algorithm 4) can resume. By randomly deleting customers and recomputing the values of the tables (or cluster centers) noise is added to the process, thus reducing the risk of over- or under-fitting a solution to the problem at hand. These steps are the foundation of the model described as the Time-varying Dirichlet Process Mixture Model (TVDPM) by Caron et al. (2007). However, in this report we refer to the process of deleting customers and moving tables (described above) as DDP(·) (short for Dependent Dirichlet Process).

4.8 Sub-conclusion

We conclude this chapter by listing the achievements gained by introducing the Dirichlet Process (DP).

As described in the introduction to this chapter, the goal has been to find a suitable process that can be used in iterative nonparametric Bayesian analysis as a prior distribution. One of the main problems in determining a suitable nonparametric prior distribution is finding a process where the complexity of the model is learned by the data itself and not specified a priori. As noted in the introduction, the DP is ideally suited for this purpose. The DP is an example of a nonparametric prior. By working nonparametrically we allow for modeling arbitrarily complex distributions in a computationally tractable manner. Furthermore, as mentioned in Section 4.6, inference and prediction with the DP can be done using Markov Chain Monte Carlo simulation techniques as Gibbs sampling which further adds to the computational tractability of the DP.

The DP is as a so-called infinite mixture model with clustering properties. This means that it is able to cluster data samples in a potentially unbounded number of clusters (the nonparametric approach). In Section 4.4 we described the foundation of the DP based on the infinite mixture model and then expanded it to the Dirichlet Process Mixture Model in Section 4.4.1. In Section 4.6 we then described the algorithm Chinese Restaurant Process (CRP) which is a sampling scheme that implements the DP. In the CRP illustration of fig. 4.7 we explicitly showed the clustering property of the DP and how clusters of the distribution are drawn from an infinite set of clusters. In Section 4.7, we concluded the chapter by expanding the (stationary) DP to deal with time-series data. In order to preserve the randomness of the DP distribution, two steps of randomly deleting clusters and data samples and moving clusters has to further be executed when moving from one time step to another.

In conclusion, the DP described in this chapter meets the requirements as a nonparametric prior as described in the introduction of this chapter. In Chapter 6 we will describe the use of the DP in conjunction with Hierarchical Nonparametric Belief Propagation (HNBP). We first start by describing Nonparametric Belief Propagation in Chapter 5.
Chapter 5

Nonparametric Belief Propagation

The purpose of this chapter is to describe the Nonparametric Belief Propagation (NBP) framework by Sudderth et al. (2003). NBP is a framework for performing computationally tractable belief propagation using MCMC simulation in undirected graphs (i.e. Markov networks) based on a nonparametric approach. The NBP framework is the foundation for the Hierarchical NBP framework of Chapter 6. This chapter is based on the description of NBP by Sudderth et al. (2003), but the conclusions, interpretations and illustrations are those of this author. Furthermore, the chapter is technically detailed; describing the fundamental algorithms of the NBP framework. The chapter concludes with a discussion of the shortcomings of NBP in relation to filling a role in an MPF as defined in Section 2.5.7.

5.1 Introduction

Nonparametric Belief Propagation is an example of how the belief propagation paradigm, using stochastic simulation, can be implemented in a graphical model. The algorithm was developed by Sudderth et al. (2003)\(^1\) and aims to combine techniques from particle filtering and Belief Propagation (as originally described by Pearl (1988)). As mentioned earlier, exact belief propagation is generally computationally intractable for real world problems because of the occurrence of complex integrals. However, belief propagation is sound as a general concept of how beliefs can be passed between nodes of a graph. NBP offers an alternative framework for computing beliefs and passing messages between nodes based on approximation and nonparametric methods with similar semantics to that of Belief Propagation.

The engine of belief propagation in NBP is message passing (as it is in traditional belief propagation). A message sent from one node to another is the product of all incoming

\(^1\)Developed in parallel to NBP, PAMPAS by Isard (2003) is an almost identical algorithm. In this paper we use notation from both NBP and PAMPAS but the algorithm described in this chapter is the NBP algorithm by Sudderth et al. (2003).
messages and local evidence. This product is potentially computationally intractable as well but NBP implements approximation of this message product by using Gibbs sampling.

The NBP framework uses an undirected graph (Markov network) to represent node relationships. Instead of using conditional probability distributions to describe the relationship between nodes (as in Bayesian networks), NBP uses potential functions to describe the compatibility between nodes. Potential functions are not necessarily probability distributions, but are more general functions specific to the application. Often NBP is deployed in vision applications and so the nodes of the network represent features present in the image. In this context, compatibility potential functions between nodes are simply distances between features in the image in the cartesian coordinate system. Although compatibility potentials are not necessarily proper probability distributions, the normalized product of all potential functions in the graph equates to the joint probability distribution of all random variables in the graph.

NBP can be considered a generalized particle filter. Traditional particle filters are optimized for temporal sequence processing and therefore have graphs that essentially are simple Markov chains (as shown in fig. 5.1.A). In general, particle filters perform sequential Bayesian analysis by propagating sets of particles that approximate the true posterior probability distributions between each time step of the chain. NBP borrows from particle filters by reusing the strategy of particle-based approximation techniques and expanding the 1D Markov chain (fig. 5.1.A) into a 2D Markov network (fig. 5.1.B+C). This way, NBP is able to account for the level of complexity present in real world problems, by extending traditional particle filters to a more generalized form, applicable to general undirected graphs. This includes graphs with loops (see fig. 5.1.B). When performing Bayesian inference using NBP in graphical models with loops, there is no guarantee that the beliefs will converge towards their true posterior distributions. This problem is identical to that
Figure 5.2: Example of NBP network with four NBP nodes. Node \( s_i \) is a wrapper for two nodes, \( x_i \) and \( y_i \). Each node \( s_i \) is connected to an element of the receptive field (in this case a \( 2 \times 2 \) matrix). New evidence is introduced into the network through \( y_i \) as defined by \( \phi_i(x_i) \).

seen when applying Pearl’s Belief Propagation algorithm to loopy Bayesian networks, known as Loopy Belief Propagation. However, as mentioned earlier, there is good empirical evidence that Loopy BP will in fact produce useful results (Sudderth et al. 2003, Freeman & Pasztor 1999, Sun, Shum & Zheng 2002).

5.2 Undirected Graphical model

As mentioned earlier, the graph of the NBP framework is an undirected graphical model. The graph \( G \) consists of a set of nodes \( V \) and corresponding edges \( E \). Each NBP node \( i \in V \) consists of a hidden node \( x_i \) and a local observation node \( y_i \), so that \( \{i|(x_i, y_i)\}_{i=1}^N \), where \( N \) is the number nodes in \( V \). Furthermore, the neighborhood of NBP node \( i \) is defined as \( \text{ne}(i) \triangleq \{j|(i, j) \in E\} \), where \( j \in V \), is a node in the neighborhood of \( i \). An example of a NBP graph with four nodes can be seen in fig. 5.2.

As mentioned earlier, \( G \) does not encode node relationships as conditional probability distributions as in a Bayesian network, but instead as potential functions. We generally distinguish between two types of potential functions in \( G \). The first type defines the relationship between two neighboring \( x \) nodes, \( x_i \) and \( x_j \), as \( \psi_{i,j}(x_i, x_j) \). Generally, potential functions can be interpreted as the “difference” or “compatibility” between the two nodes. In vision applications we often define \( \psi_{i,j} \) as simply a distance in the Euclidean space (often 2D or 3D). The second type is the potential function \( \phi_i(x_i) \) between node \( x_i \) and \( y_i \) inside NBP node \( i \). The role of this potential function is to define how input from node \( y_i \) (the receptive field) is introduced into the network at \( x_i \).
The two aforementioned potential functions can be used to define the joint probability distribution of all the nodes in $G$ as:

$$p(x, y) = \alpha \prod_{(i,j) \in E} \psi_{i,j}(x_i, x_j) \prod_{i \in V} \phi_i(x_i),$$

(5.1)

where $\alpha$ is a normalizing constant that ensures that $p(x, y)$ is a proper probability distribution.

### 5.3 Belief Propagation

As new evidence is introduced into the network, the nodes of the graph $G$ form new beliefs by propagating messages between the nodes. A message sent from NBP node $i$ to $j$, informs $j$ of what state node $i$ is expecting $j$ to be in. Node $j$ can then combine this information with messages coming from other connected nodes in the neighborhood of $j$. The computation of a message from NBP node $i \in V$, to $j \in ne(i)$ at iteration $n$, denoted as $m^n_{i,j}$, can be computed as:

$$m^n_{i,j}(x_j) = \alpha \int_{x_i} \psi_{i,j}(x_i, x_j) \phi_i(x_i) \prod_{k \in ne(x_i) \setminus x_j} m^{n-1}_{k,i}(x_i) dx_i$$

(5.2)

Here, $ne(x_i) \setminus x_j$ denotes the neighborhood of hidden node $x_i$ excluding $x_j$ and $\alpha$ is a normalizing constant. We see that the message $m^n_{i,j}$ consists of the product of all incoming messages to node $i$ (the $\prod_k m^{n-1}_{k,i}(x_i)$ term, known as the message product\(^2\)), multiplied with the local evidence $\phi_i$ and the compatibility function between $i$ and $j$, $\psi_{i,j}$.

After the newly introduced evidence has been propagated between the nodes of $G$ using eq. 5.2, we then can compute the belief of each of the nodes in $G$. The belief of node $i$ is the posterior distribution $p(x_i|y)$. The belief is based on all incoming messages as well as the local evidence introduced by that node. We therefore omit the subscript on $y$ to show that the belief of a node is based, not only on local evidence, but on all neighborhood evidence. In NBP this posterior is computed as an approximation $\hat{p}$:

$$\hat{p}^n(x_i|y) = \alpha \phi_i(x_i) \prod_{j \in ne(x_i)} m^n_{j,i}(x_i)$$

(5.3)

When $G$ is a tree, the approximate posterior $\hat{p}^n(x_i|y)$, converges to the true posterior $p(x_i|y)$, but when $G$ is arbitrary and contains cycles, there is no guarantee of convergence (as in Loopy Belief Propagation, see Section 5.1).

\(^2\)The term 'message product' is defined slightly different in the original paper by Sudderth et al. (2003), as: $\phi_i(x_i, y_i) \prod_k m^{n-1}_{k,i}(x_i)$. However, in this report, the message product is defined only as $\prod_k m^{n-1}_{k,i}(x_i)$.
5.3.1 Nonparametric Representation

A cornerstone of NBP is the nonparametric representation of distributions. We motivate the use of nonparametric models in the following. For instance, say we wanted to evaluate eq. 5.2. This would require determining a closed-form solution for the integral of eq. 5.2. However, this task is only feasible for simple applications. When the potential functions become multi-modal and high-dimensional (as is often the case in realistic applications) analytical evaluation is no longer feasible. One solution could simply be to use a simple parametric potential function distributions, but this restricts the flexibility of the model (and thereby its ability to model complex relationships). By using nonparametric modeling we can address this issue in one elegant solution. In nonparametric modeling there is no a priori assumption of the structure of the model, instead, the complexity of the model is determined by the data. This strategy has been adopted in NBP. Each message distribution is instead represented nonparametrically as a *kernel density estimate* of a mixture of \( M \) weighted Gaussians:

\[
m_{i,j}(x_i) = \sum_{n=1}^{M} w_i^{(n)} \mathcal{N} \left( x_i; \mu_i^{(n)}, \Lambda_i^{(n)} \right),
\]

(5.4)

where \( w \) is a weight, \( \mathcal{N} \) denotes the Gaussian distribution with mean \( \mu \) and variance \( \Lambda \). Eq. 5.4 shows how a message distribution is determined nonparametrically as a *convex combination* of \( M \) Gaussian kernels (as the NBP graph is a generalized particle filter we often denote these weighted kernels as *particles*). By adding more particles we can create more complex distributions and vice versa. This means that we can obtain arbitrarily precise approximations of distributions by adjusting \( M \) (even obtaining the exact distribution by letting \( M \to \infty \)). As \( M \) is adjustable, the system designer can chose an \( M \) that 1) fits the application at hand, 2) fits the resources available. For an example of a nonparametric representation of a distribution refer to fig. 5.3.

5.3.2 Message Propagation

According to eq. 5.2, the forming of message \( m_{i,j} \) consists of gathering the messages from the nodes in \( i \)’s neighborhood \((k \in \text{ne}(i) \backslash j)\), and multiplying their product with the local evidence from \( i \). Fig. 5.4 is an illustration of this process. At the far right of the illustration, nodes \( k \) of \( i \)’s neighborhood can be seen. Messages from these nodes are received by \( i \) and multiplied to form the message product. Hereafter, the message product is multiplied with the local evidence from \( i \), \( \phi_i \), and the compatibility function between \( i \) and \( j \), \( \psi_{i,j} \). Then \( m_{i,j} \) is then formed after integrating the resulting product using Monte Carlo integration as we will see later in Section 5.3.5.

The illustration further shows the message product complexity. Given that each of the messages from all \( k \) is a mixture of \( M \) weighted Gaussians (according to eq. 5.4), and that there are \( k \) messages in \( i \)’s neighborhood, a total of \( O(M^k) \) multiplications are (worst case)
needed to compute the message product. As the number of nodes in $G$ increases (i.e. $k$ increases) the number of computations grows exponentially. This implies that computing eq. 5.2 becomes intractable for large networks. This problem is further described in the following section.

### 5.3.3 Complexity of Message Product

As mentioned in the previous section, computing the integral of eq. 5.2 can be computationally intractable for large networks. The root of the problem is the message product of eq. 5.2, where all incoming messages to node $i$ are multiplied together (repeated here):

$$\Psi_{ne(x_i)\setminus x_i}(x_i) \triangleq \prod_{\forall k \setminus j} m_{k,i}(x_i) \quad (5.5)$$

The dimensionality of $\Psi$ is exponential given that $m_{i,j}^n$ consists of $k$ messages each having $M$ Gaussian kernels (based on the nonparametric representation of messages in eq. 5.4). Computing the product of $k$ messages with each $M$ kernels would require $O(M^k)$ operations to compute $m_{i,j}^n$. 

---

Figure 5.3: Example of a nonparametric particle based representation of a density. Thin black lines represent particles (in the NBP case particles are Gaussian kernels with varying means and variances). Thick red line represents the resulting density as the sum of the particles (the convex combination) (eq. 5.4). With a nonparametric particle based representation we can create arbitrarily complex multi-modal densities without having to know their closed-form expressions.
To ensure computational tractability, NBP instead approximates the product density by drawing $M$ samples from the message product in $O(knM^2)$ operations where $n$ is the number of iterations in the sampling process (the more iterations the higher precision). The sampling process is therefore an approximation to the true message product density. However, at the price of precision we obtain computational tractability as NBP message passing scales well with increasing network sizes. NBP ensures this tractability by approximating the message product in eq. 5.5 as a mixture of $M$ weighted Gaussian kernels, as in eq. 5.4, by using the Gibbs sampler first described in Section 3.6.

5.3.4 Approximating the Message Product

If the message product of eq. 5.5 is seen as a joint probability distribution of all of the components of the incoming messages multiplied (incoming message components (kernels) corresponding to conditional probability distributions) then it is possible to approximate this joint probability distribution using a Gibbs sampler. In this section we proceed with an in-depth description of the application of the Gibbs sampler to the approximation of the message product.

Let, the *label* $l_j$ denote a specific kernel ($l_j \in [1 : M]$), in the $j$’th message distribution ($j \in [1 : d]$, where $d$ is the number of messages in the message product). In other words, $j$ is
a message being sent from node $t$ in the neighborhood of node $s$, $t \in \text{ne}(s)$, to $s$, consisting of a mixture (set) of Gaussians. In NBP, the Gibbs sampling process can be described as (please note that the $k$, $j$ and $i$ in the following are not related to their instances in the previous sections):

1. **Initialize**
   Run through all the messages $j$, involved in the message product. For each $j$, a label $l_j = i$ is stochastically selected proportional to the weight of the label, $w_j^{(i)}$.

2. **Start sample process**
   (a) For each message $j$, a new set of weights $\{w_j^{(i)}\}_{i=1}^M$ are computed, while conditioning on the remaining set of fixed labels $\{l_k\}_{k \neq j}$. The new weights are computed based on their prior values and the values of the conditioning labels.
   (b) Select a new label $l_j$ based on the newly computed weights and the conditioned labels $\{l_k\}_{k \neq j}$ by sampling from $p(l_j = i) \propto w_j^{(i)}$.
   (c) This process continues for $n$ iterations. As the process iterates, each kernel’s, or particle’s, weight is adapted continuously. As more iterations are completed, some particles grow “stronger” (i.e. weights increase) and become more and more likely to be selected in the stochastic selection process of next iteration.

3. **Post sample process**
   (a) After the end of the sampling process, the last selected set of labels $\{l_j\}_{j=1}^d$ of Step 2, will likely be the strongest particles (although not with certainty because of the stochastic nature of the selection process). These $d$ labels then represent all the $M^d$ labels of the messages.
   (b) Compute mean $\bar{\mu}$ and variance $\bar{\Lambda}$ from: $\prod_{j=1}^d \mathcal{N}(x; \mu_j^{(l_j)}, \Lambda_j^{(l_j)})$.
   (c) Draw one sample $\hat{x} \sim \mathcal{N}(x; \bar{\mu}, \bar{\Lambda})$. This sample is one representative sample for the message product and the sought-after Gibbs sampling approximation eq. 5.5.

In more compacted pseudo-code, the Gibbs sampling implementation can be described as shown in Algorithm 5.

**5.3.5 NBP Message Construction Algorithm**

In Section 5.3.4, we addressed the issue of determining the message product $\Psi$ of eq. 5.5. In this section we conclude our description of NBP by describing the algorithm for constructing new messages (Algorithm 6) using the message product approximation (i.e. completing computation of eq. 5.2). In this report we follow the algorithm described by Sudderth et al. (2003) (with added detail for clarification beyond that in Sudderth et al.)
Algorithm 5 NBP Gibbs sampling of message product (simplified)

1: # 1. Initialize:
2: for each message \( j \in [1 : d] \) do
3: \quad Select a label \( l_j \), by sampling uniform r.v. \( p(l_j = i) \propto w_j^{(i)} \)
4: end for
5: # 2. Start sampling:
6: for each iteration \( n \) do
7: \quad for each message \( j \in [1 : d] \) do
8: \quad \quad Compute new set of weights \( \{w_j^{(i)}\}_{i=1}^M \)
9: \quad \quad Select a new label \( l_j \), from \( p(l_j = i) \propto w_j^{(i)} \)
10: \quad end for
11: end for
12: # 3. Post sampling:
13: Compute mean \( \bar{\mu} \) and variance \( \bar{\Lambda} \) from \( \prod_{j=1}^d \mathcal{N}(x; \mu_j^{(l_j)}, \Lambda_j^{(l_j)}) \)
14: Return one sample: \( \hat{x} \sim \mathcal{N}(x; \bar{\mu}, \bar{\Lambda}) \)

(2003)), however, we can recommend studying the similar algorithm PAMPAS by Isard (2003). The PAMPAS algorithm only differs in minor areas and furthermore, a fully working Matlab implementation is available (Sigal 2003) based on Ihler fast and expert Matlab/C++ KDE toolbox implementation (Ihler 2003).

The construction of a message in NBP is a four step operation. The construction process is overall a Monte Carlo integration of eq. 5.2, using importance sampling (described in Section 3.7) with an embedded Gibbs sampling procedure (Algorithm 5) to draw samples from the message product. The objective is to send a message \( m_{i,j} \) from node \( i \) to \( j \). We assume we have the following input messages to NBP node \( i \): \( m_{n-1}^{k,i}(x_i) \), \( \forall k \in \text{ne}(x_i) \setminus x_j \). The construction process for the output message \( m_{i,j} \) is shown in Algorithm 6. Note that in Algorithm 6 we have assumed that the potential function between two nodes \( x_i \) and \( x_j \), \( \psi_{i,j} \), represents the difference between the two nodes. In more generic cases we refer to Sudderth et al. (2003).

5.4 Sub-conclusion

Nonparametric Belief Propagation (NBP) by Sudderth et al. (2003) is a generalized particle filter that offers belief propagation in undirected graphical models (e.g. Markov networks) with arbitrary graph topologies using stochastic simulation. Section 5.3 describes how belief propagation is performed in NBP.

Connections between nodes in NBP are modeled as potential functions, analogous to conditional probability distributions in directed graphs. NBP nodes of the graph implement each two variables, a hidden node \( x \) and a receptive node \( y \). Messages are passed
Algorithm 6 NBP Message Construction

1. \# 1. Using Algorithm 5, draw \( M \) equally weighted message product samples:
   2. \textbf{for} each \( n \in M \) \textbf{do}
      3. \( \tilde{x}_i^{(n)} \sim \Psi_{ne(x_i) \setminus x_j}(x_i) \)
      4. \( w_i^{(n)} = 1/M \)
   5. \textbf{end for}

6. \# 2. Apply importance correction to particle weights using local likelihood potential:
   7. \textbf{for} each \( n \in M \) \textbf{do}
      8. \( w_i^{(n)} = w_i^{(n)} \phi_i(x_i = \tilde{x}_i^{(n)}) \)
   9. \textbf{end for}

10. \# 3. Propagate samples to recipient neighbor node \( j \) through potential function:
   11. \textbf{for} each \( n \in M \) \textbf{do}
      12. \( \tilde{x}_j^{(n)} \leftarrow \psi_{i,j}(x_i = \tilde{x}_i^{(n)}, x_j) \)
   13. \textbf{end for}

14. \# 4. Assemble message \( m_{i,j} = \{\mu_{i,j}^{(n)}, \Lambda_{i,j}^{(n)}, w_{i,j}^{(n)}\}_{n=0}^M \) as:
   15. \textbf{for} each \( n \in M \) \textbf{do}
      16. Set \( \mu_{i,j}^{(n)} = \tilde{x}_j^{(n)} \)
      17. Choose \( \Lambda_{i,j}^{(n)} \) according to Ihler et al. (2004, p. 7) or Silverman (1986).
      18. Set \( w_{i,j}^{(n)} \) equal to importance weights generated in \textit{Step 4}.
   19. \textbf{end for}
between neighboring hidden nodes in order to form new beliefs in the graph (as defined in eq. 5.2) with similar semantics to that of Belief Propagation by Pearl (1988). All distributions (messages and beliefs) are represented nonparametrically. This means that arbitrarily complex distributions can be represented without having to compute their closed-form expressions.

Beliefs of nodes are computed by marginalizing the hidden variable of the node, conditioned on the receptive variables in the neighborhood of the node, according to eq. 5.3. However, as described in Section 5.3.3, computation of messages can prove intractable. Therefore, NBP implements a Gibbs sampling procedure to approximate message products to avoid exponential growth in computation time as the number of nodes in a node’s neighborhood increases. Lastly, the computation of a new message is done using an importance sampler with an embedded Gibbs sampler to draw samples from the message product. The importance sampler performs the Monte Carlo integration needed to compute eq. 5.2.

Despite the many good features of NBP, it lacks to implement especially two important features described as relevant for an MPF in Section 2.5.4:

1. Representation of causes in many layers. In the NBP network described by Sudderth et al. (2003), hidden variables in each NBP node provide abstraction of the receptive variables. However, in an MPF, many layers of abstraction are needed in order to be able to model high-order causes and dependencies.

2. Account for time. The NBP framework has no support for modeling objects that evolve over time.

In summary, it is necessary to expand the NBP message passing algorithm to support passing messages between different layers of scale and to implement time in the NBP algorithm. This means defining a novel extension of the “flat” NBP to a hierarchy of NBP nodes. In Chapter 6 we describe hierarchical NBP (HNBP), which is novel extension of NBP to hierarchies using Dirichlet Processes. In Chapter 7 we continue our expansion of HNBP into a temporal framework, Dynamic HNBP (DHNBP), that is able to model time dependent distributions.
Chapter 6

Hierarchical Nonparametric Belief Propagation

The purpose of this chapter is to introduce the Hierarchical Nonparametric Belief Propagation framework (HNBP). HNBP is an extension of NBP (see Chapter 5) into a multi-layered hierarchy. We first describe the key components of HNBP framework, then proceed to describe the HNBP algorithm. The chapter is then concluded with a sub-conclusion.

6.1 Introduction

In this chapter we build on the work of the previous chapter on Nonparametric Belief Propagation (Chapter 5), as developed by Sudderth et al. (2003). Furthermore, we present a novel extension of the one-layered NBP model to the multi-layered Hierarchical NBP (HNBP).

As mentioned in Section 5.4, in order to model causes in many layers of spatial scale it is necessary to be able to perform message passing between nodes of different hierarchical scale and context. Originally, NBP was developed by Sudderth et al. (2003) to work as a single layered model that directly receives input from the receptive field and then provides one layer of abstraction of the input. Sudderth et al. (2003) provided a novel algorithm for nonparametric message passing (with similar semantics to that of Pearl (1988)’s Belief Propagation algorithm) only applied to undirected Markov networks instead of directed Bayesian networks. The NBP algorithm offers computationally tractable Bayesian analysis in realistic settings that scales well with complexity. One of the main themes of NBP is the utilization of nonparametrics, where distributions are represented as a convex combination of smaller parametric parts (see also Section 5.3.1).

The goal of this chapter is to build on the foundation of the one-layered NBP framework and to improve it in three main areas:

1. Add the notion of spatial layers. As mentioned, the NBP framework only facilitates
one spatial layer of abstraction. But as described in Section 5.4, in order to be able to represent complex objects, it is necessary to be able to represent objects of many layers of spatial scale. Thus, it is necessary to add the concept of spatial layers of scale to the NBP framework and adjust the NBP algorithm accordingly.

2. Add a prior distribution to the framework in order to facilitate iterative fully Bayesian analysis (inference and prediction) using an explicit prior distribution in a nonparametric setting. Furthermore, the prior distribution has to cluster continuous input data into discrete categories, thus aiding to provide invariant representation of objects (described in Section 2.1).

3. Add the ability of parent nodes “teaching” child nodes (described in Section 2.5.4). This is done by allowing parent nodes to shape the prior distributions (the Dirichlet Process of Chapter 4) of child nodes in layers below through feedback streams (as described in Section 2.1). This way parents in layers above become “teachers” for children below.

The remainder of the chapter is divided up in three main parts. First, we describe the extensions above in detail. Hereafter, we present the updated HNBP algorithm of NBP. Lastly, we conclude the chapter by summarizing the objectives gained in extending NBP to HNBP.

6.2 HNBP Features

In this section we describe our novel extensions in HNBP made to the original NBP framework by Sudderth et al. (2003) (see also Chapter 5).

6.2.1 Hierarchy

The first extension to the NBP framework is to convert the one-layered hierarchy into a multi-layered one. In fig. 6.1 an illustration of the resulting framework can be seen. In NBP, nodes are connected laterally (as seen on the left side fig. 6.1). However, in HNBP, nodes on the same layer have no connections with each other. Thus, a node in layer $k$ can only have connections with other nodes in layer $k - 1$ or $k + 1$ and thus not with other nodes in layer $k$. This results in a taxonomic hierarchy of nodes where some nodes are “parents” of other nodes (“children”). As we ascend the hierarchy, parents accumulate the scope of their children. This results in an compression of space as the hierarchy is ascended and as (parent) nodes accumulate more and more space.

The fact that nodes in layer $k$ are not interconnected does not necessarily mean that they are independent of each other. As we described in Section 2.2.2, child nodes in layer $k$, which are connected to the same parent node in layer $k + 1$, are thus conditionally
Figure 6.1: Transformation from flat NBP network to multi-layered hierarchical NBP network (HNBP). Input nodes are shaded as gray while abstracting nodes are shaded white. Furthermore, an extra node (the Dirichlet Process, DP) has been added to each node (shaded in black). The DP can be considered a hyperparameter to the white HNBP node.

independent of each other given the parent node in layer $k + 1$. The conditional independence property means that changes in belief of a node in layer $k$ induces a change in the likelihood of the other nodes in layer $k$ that share the same parent node in layer $k + 1$. As mentioned in Section 2.2.2, this property is also known as “explaining away”.

Although we limit nodes to only have interlayer connections (intralayer connections will have another purpose and will be described in Chapter 7), we do not otherwise limit the network structure of the HNBP framework. We therefore allow for arbitrary network structures. So, per construction, the HNBP framework facilitates nonparametric belief propagation across hierarchies of scale. This extension leads to alterations in the original NBP algorithm, which we will describe in Section 6.3.3.

From fig. 6.1 and 6.2 we see that all HNBP nodes (in white) have two attached nodes:

- Input nodes: Shaded in gray
- Dirichlet Process nodes: Shaded in black (small node)

In the following section we explain the purpose of the input nodes of the HNBP network. The application of the DP in HNBP will be described in Section 6.2.3.
6.2 HNBP Features

Figure 6.2: In Chapter 5 we described that an NBP node consists of two nodes: An input variable \( y \), shaded in gray) and an abstracting node \( x \), white) with a distribution \( \phi \) that introduces the input data from \( y \) to \( x \). In HNBP we expand the concept of an NBP node to include the Dirichlet Process (DP). The DP distribution is accessed through the \( \gamma \) distribution.

6.2.2 Input Nodes

All HNBP nodes have input nodes attached. However, we distinguish between two types of input: First, all HNBP nodes in the lowest input layer (the receptive field) are all required to have an input node. These nodes introduce the main input being modeled by the HNBP network. For instance, if the HNBP is applied to a computer vision image problem, input nodes in the lowest layer would introduce data from the image. However, as can be seen in fig. 6.1, we also allow HNBP nodes in layers above the input layer (receptive field) to have inputs. These nodes however, do not introduce evidence from the main source of input (e.g. an image). Nodes in layers above the input layer exist in a different hierarchical scale and context and thus the input data from the main input source is not applicable to these nodes. Instead, the input nodes in higher layers allow for “manual tuning” of beliefs by the system designer. We motivate the use of these input nodes by referring back to the bike model of fig. 2.2. If we, for instance, imagined that the wheel node of Layer 2 is uncertain whether the input it is receiving from the nodes below in fact identifies a wheel present in the data, we could use the manual tuning-node to manually alter the belief of that node towards representing a wheel. However, the use of a manual tuning node is reserved for the system designer and is not applicable to autonomous input feeding as the nodes in the receptive field are. The manual tuning-nodes in upper layers are therefore optional, whereas input nodes in the receptive field are mandatory. The manual tuning-nodes for all high-layer nodes can therefore be left out if they are not needed, but have been included in all following figures and equations for completeness.

6.2.3 Causal Feedback and the Dirichlet Process

As described in the goal specification of Section 2.5, an important goal of the master thesis project is to expand the NBP algorithm to support hierarchical causal feedback to shape prior distributions, thus allowing parent nodes to become “teachers” for child nodes.

By expanding NBP to a hierarchy, the notion of layers of hierarchical scale is added
Figure 6.3: Canonical HNBP network. Three layers are presented. Each node $x_k$ has an associated Dirichlet process $\gamma_k(x_k)$ and an input variable $y_k$. Furthermore, $m_{l,k}$ is the message passed from node $x_l$ to node $x_k$, $\psi_l(x_l, x_k)$ is the potential function between $x_l$ to node $x_k$, and finally, $\phi_k(x_k)$ is the potential function that models the input from node $y_k$ to node $x_k$.

Parent nodes in layer $k+1$ abstract child nodes in layer $k$ and accumulate the scope of their child nodes. This means that parent nodes span over a greater scope of space (and time, as we shall see in the DHNBP model of Chapter 7) and has a greater overall “understanding” than their children. Thus, if ambiguities in the beliefs of child nodes occur, parent nodes can disambiguate and “teach” child nodes by shaping their beliefs and “explain away” (see Section 2.2.2). This is possible, because parent nodes are likely to receive input from several child nodes and so their beliefs are based on several sources. Thus, if one child node delivers inconclusive evidence, the parent node can feedback information to that child node, based on joint evidence from the other (likely) more certain child nodes, and correct it.

Based on the discussion of this section, the objective has been to design a framework and algorithm where parent nodes can “teach” child nodes. Lee & Mumford (2003) further describes the concept as: “feedback from higher inference areas provides the priors to shape the inference at earlier” layers. So more specifically, the objective has been to design a framework where prior distributions of child nodes can be shaped by feedback from parent nodes. For this purpose, the Dirichlet Process (DP) is ideally suited (see also Chapter 4).
The role of the DP in HNBP is to cluster feedback data streams from parent nodes, thus aiding to provide invariant representations. In fig. 6.3 a canonical HNBP network can be seen. Each node of fig. 6.3 has a DP associated with it. The distribution of each DP is determined by the Chinese Restaurant Process (CRP, Section 4.6). The DP distribution for HNBP node $x_k$ is accessed through the $\gamma_k(x_k)$ distribution and serves as prior distribution for $x_k$. Thus, the prior distribution $\gamma_k(x_k)$ is shaped by feedback data streams (causal messages) from the parents $x_l$ and $x_m$ of node $x_k$. Updating $\gamma_k(x_k)$ with causal messages is then done using the CRP algorithm of Section 4.6. So the process of updating the DP consists of drawing samples from the causal message product ($\Psi_{\text{pa}(x_k)}$) using the Gibbs sampler (Algorithm 5 of Section 5.3.3) and then clustering these samples using the CRP algorithm (Algorithm 4 of Section 4.6). Technically, the updated DP $\gamma_k$ can be considered a posterior DP distribution, however, we reserve the term “posterior” for the computation of the belief distribution of node $x_k$ (further described in Section 6.3.2).

6.2.4 Bayesian Analysis

The four step process of Bayesian inference and prediction in HNBP is built on the similar two step process of NBP. In NBP, Bayes’ theorem (eq. 4.1) can computed in two steps:

1. Messages are passed between nodes
2. Once message passing is complete, the belief of the nodes can be computed

In HNBP we introduce two more steps at the beginning and at the end:

1. Prior distributions (DPs) are updated using causal feedback streams from parents
2. Messages are passed between nodes
3. Once message passing is complete, the belief of the nodes can be computed
4. The DP is “overwritten” with the belief of the node, thus completing the Bayesian analysis

The four steps of the HNBP algorithm amounts to one complete application of Bayes’ theorem. Thus, when new evidence is available, the above four step algorithm is activated and ends with the overwriting procedure of the DP. Then as new evidence again becomes available the above algorithm is executed again and so on. In Section 6.3.3 we further describe this process in detail when we present the HNBP algorithm.

6.2.5 Invariance in HNBP

The objective of this thesis has been to build a model that is able to represent invariant features that generalizes well without over-fitting solutions in the Bayesian analysis. In order to avoid over-fitting solutions, it is necessary to introduce noise into the system by
6.3 HNBP Algorithm

In this section we describe the HNBP algorithm itself. The algorithm can be considered an extended version of the NBP message construction algorithm (Algorithm 6) in Section 5.3.5. Thus, in HNBP we reuse the NBP Gibbs sampling of the message product (Algorithm 5), but we present an updated HNBP message construction algorithm to the NBP message construction algorithm (Algorithm 6).

The resulting HNBP algorithm can be divided up into two phases: In Phase One, the DP of each node is updated based on the causal messages (messages from parents). We denote the CRP updating process as: \( \gamma_k(x_k) \leftarrow \text{CRP} \). After updating the DP, message passing between nodes commence. When message passing is complete, Phase Two can be initiated. In Phase Two, the beliefs of all nodes are computed and the “overwrite” operation of the DPs with the newly computed beliefs of the nodes occurs. This completes one round of Bayesian analysis. As new evidence becomes available in the network, Phase One and Two are executed again.

6.3.1 Phase One: HNBP Message Construction

We denote the message product of causal messages:

\[
\Psi_{pa(x_k)}(x_k) \triangleq \prod_{\forall n \in pa(x_k)} m_{n,k}(x_k), \quad (6.1)
\]

1Sentence paraphrased from Dean (2006) (originally Ullman & Soloviev (1999)).
6.3 HNBP Algorithm

where \( \text{pa}(x_k) \) denotes the parents of node \( x_k \). Using Algorithm 5, we can draw \( M \) samples from \( \Psi_{\text{pa}(x_k)} \) as:

\[
\{ S_n \}_{n=0}^{M} \sim \Psi_{\text{pa}(x_k)}(x_k)
\]  

(6.2)

The samples drawn in eq. 6.2 from the message product, are then used to update the prior distribution (the DP) using the CRP algorithm (Section 4.6):

\[
\gamma_k(x_k) \leftarrow \text{CRP}(\gamma_k(x_k), \{ S_n \}),
\]  

(6.3)

Thus, the samples \( \{ S_n \}_{n=0}^{N} \) drawn from the causal message product are augmented with the current state of the DP \( \gamma_k(x_k) \). The state of the DP has thus been updated with the most recent available causal data.

The next step is for the node to send out new messages. The “sharpened” DP is now used to reweight the messages being sent out. Reweighting is identical to the process of predicting samples from the DP, as described in Section 4.6.5. Referring to fig. 6.3, we can thus compute a message from node \( x_k \) to \( x_j \) as:

\[
m_{k,j}(x_j) \leftarrow \alpha \int_{x_k} \phi_k(x_k)\gamma_k(x_k)\psi_k(x_k, x_j) \prod_{\forall n \in \text{ne}(x_k) \setminus j} m_{n,k}(x_k) dx_k,
\]  

(6.4)

where \( \text{ne}(x_k) \setminus x_j \) denotes the set of neighbors above and below node \( x_k \) excluding node \( x_j \). Eq. 6.4 is based on the original NBP message (eq. 5.2) and only differs in the inclusion of the DP distribution \( \gamma_k \). As with the NBP algorithm (see Section 5.3.4), the message product, \( \prod m_{n,k}(x_k) \), of eq. 6.4 is approximated using Algorithm 5.

This concludes Phase One of the HNBP algorithm. Phase One is repeated until beliefs are believed to have converged. Hereafter, Phase Two can be executed and beliefs can be computed. We now finish our description of the HNBP algorithm by describing Phase Two.

6.3.2 Phase Two: HNBP Belief Update

In Phase Two of the HNBP algorithm, we compute the belief of HNBP node \( x_k \) (assuming that enough message passing has occurred in Phase One, for beliefs to have reached a steady state). Again, referring to fig. 6.3, the belief of node \( x_k \) can be computed by combining the total message product \( \Psi_{\text{ne}(x_k)} \) (both nodes above and below node \( k \)) with the input evidence introduced to the node itself \( \phi_k(x_k) \) (if applicable) and the DP \( \gamma_k(x_k) \):

\[
b_k(x_k) = \alpha \gamma_k(x_k)\phi_k(x_k) \prod_{\forall j \in \text{ne}(x_k)} m_{j,k}(x_k)
\]  

(6.5)

Eq. 6.5 is an extension of the NBP belief computation (eq. 5.3). The main differences between these two equations are the inclusion of the DP and the option for input at all
layers in eq. 6.5. Note that the DP $\gamma_k(x_k)$ is again used to reweight and sharpen the belief of the node.

After computing the belief, the local DP is “overwritten” with the newly computed belief:

$$\gamma_k(x_k) = \text{CRP}(b_k(x_k))$$  \hspace{1cm} (6.6)

As shown in eq. 6.6, the DP is “overwritten” (denoted by equal sign) with the newly computed belief distribution of eq. 6.5. Note that this belief distribution is converted to a DP distribution (as denoted by “\text{CRP}(\cdot)”) before overwriting. This way, the DP is updated from a prior distribution to a posterior distribution. The DP can then be used for another round of computation as new evidence arrives. In this new round of computation, the DP will be a prior distribution and then updated to become a posterior, and so on. Note, Phase Two of the HNBP algorithm is mandatory before moving on to a new round of computation (i.e. starting over with Phase One).

6.3.3 Algorithm

In this section we provide pseudo-code versions of the two phases of HNBP (Phase One and Two) in Algorithms 7 and 8. At any iteration of Phase One, a message from node $x_k$ to $x_j$ can be constructed as in Algorithm 7. When having iterated over Phase One sufficiently, so that beliefs have converged, Phase Two for HNBP node $k$ can be computed as in Algorithm 8.

6.4 Sub-conclusion

In this chapter we described the Hierarchical Nonparametric Belief Propagation framework (HNBP). The HNBP framework is a novel extension of the NBP framework by Sudderth et al. (2003) from a one-layered (flat) hierarchy into a multi-layered hierarchy. With this extension, the important goal of facilitating bidirectional hierarchical nonparametric belief propagation has been achieved (described in Section 2.5.7). With the HNBP framework we have described a method for passing messages (both inference and prediction) between layers of different scale.

In this chapter we furthermore described the application of the Dirichlet Process (DP) in the HNBP framework. The DP serves as a nonparametric prior where messages from parent nodes are “stored”. This information is then used to shape the belief of the child node in the next iteration of computation. The implementation of the DP into the HNBP framework completes the important goal (described in Section 2.5.7) of allowing parent nodes to shape the beliefs of child nodes using recurrent feedback streams.

The HNBP framework is able to represent non-over-fitting invariant features. This ability is based on the expressiveness of the HNBP network topology (i.e. facilitating
Algorithm 7 Phase One: HNBP Message Construction of message $m_{k,j}$

1: # 1. Using Algorithm 5, draw $M$ causal message product samples:
2: for each $n \in M$ do
3: \[
\{S_n\}_{n=0}^{M} \sim \Psi_{pa(x_k)}(x_k)
\]
4: end for
5: # 2. Update the DP distribution with the samples drawn:
6: $\gamma_k(x_k) \leftarrow \text{CRP}(\gamma_k(x_k), \{S_n\})$
7: # 3. Using Algorithm 5, draw $M$ equally weighted message product samples:
8: for each $n \in M$ do
9: \[
\hat{x}_k^{(n)} \sim \Psi_{ne(x_i)\setminus x_j}(x_i)
\]
10: $w_k^{(n)} = 1/M$
11: end for
12: # 4. Apply importance correction to particle weights using local likelihood potential:
13: for each $n \in M$ do
14: \[
w_k^{(n)} = w_k^{(n)} \phi_k(x_k = \hat{x}_k^{(n)})
\]
15: end for
16: # 5. Reweight (predict) using the Dirichlet Process (see also Section 4.6.5):
17: for each $n \in M$ do
18: \[
w_k^{(n)} = w_k^{(n)} \gamma_k(x_k = \hat{x}_k^{(n)})
\]
19: end for
20: # 6. Propagate samples to recipient neighbor node $j$ through potential function:
21: for each $n \in M$ do
22: \[
\hat{x}_j^{(n)} \leftarrow \psi_{i,j}(x_i = \hat{x}_i^{(n)}, x_j)
\]
23: end for
24: # 7. Construct message $m_{k,j} = \{\mu_{k,j}^{(n)}, \Lambda_{k,j}^{(n)}, w_{k,j}^{(n)}\}_{n=0}^{M}$ as:
25: for each $n \in M$ do
26: Set $\mu_{k,j}^{(n)} = \hat{x}_j^{(n)}$
27: Choose $\Lambda_{k,j}^{(n)}$ according to Ihler et al. (2004, p. 7) or Silverman (1986).
28: Set $w_{k,j}^{(n)}$ equal to importance weights generated in Step 18.
29: end for
Algorithm 8 Phase Two: HNBP Belief Update of node $x_k$

1: # 1. Using Algorithm 5, draw $M$ equally weighted message product samples:
2: for each $n \in M$ do
3: \[ \hat{x}_k^{(n)} \sim \Psi_{ne(x_k)}(x_k) \]
4: \[ w_k^{(n)} = 1/M \]
5: end for
6: # 2. Apply importance correction to particle weights using local likelihood potential:
7: for each $n \in M$ do
8: \[ w_k^{(n)} = w_k^{(n)} \phi(x_k = \hat{x}_k^{(n)}) \]
9: end for
10: # 3. Reweight (predict) using the Dirichlet Process (see also Section 4.6.5):
11: for each $n \in M$ do
12: \[ w_k^{(n)} = w_k^{(n)} \gamma_k(x_k = \hat{x}_k^{(n)}) \]
13: end for
14: # 4. The belief of node $x_k$ is then the set of particles:
15: \[ b_k(x_k) = \{ \hat{x}_j^{(n)}, w_k^{(n)} \}_{n=0}^M \]
16: # 5. “Overwrite” the DP of node $x_k$ with the new converted belief distribution:
17: \[ \gamma_k(x_k) = \text{CRP}(b_k(x_k)) \]

overlapping receptive fields) and by learning features from multiple examples, but also on the clustering properties of the DP and how beliefs are distributed across hierarchical scales.

In the next chapter (Chapter 7) we expand the HNBP framework to model time. In Section 2.5.2 it was described that in order to create spatiotemporal representations (i.e. represent objects that co-exist in space and time) it is necessary to expand the spatial hierarchical model to facilitate the notion of time. The resulting model in Chapter 7 has been named the Dynamic Hierarchical Nonparametric Belief Propagation (DHNBP).
Chapter 7

Dynamic Hierarchical Nonparametric Belief Propagation

In this chapter, the Dynamic Hierarchical Nonparametric Belief Propagation (DHNBP) framework is described. DHNBP is our novel extension of the (equally novel) stationary HNBP framework into time, thus facilitating sequential Bayesian analysis and the ability to represent invariant spatiotemporal sequences of features.

The resulting DHNBP framework is a result of research done by this author in the area of combining computationally tractable models with the core concepts of the Memory-prediction Framework (MPF). Therefore, the resulting DHNBP model should be considered a theoretical contribution to the specific area of research.

A model such as DHNBP should ideally be able to both represent and learn sequences of features. However, in this master thesis it was only possible to focus on creating a representational model due to the time constraint of the project. Thus, in this chapter we describe how representations of invariant spatiotemporal sequences of features can be modeled, however we do not describe how they can be learned.

7.1 Introduction

As mentioned in Section 2.5.2, an important part of the Memory-prediction Framework (MPF) is its ability to represent features in space and time. According to Hawkins & Blakeslee (2004), an important property of the human intellect is its ability to correlate objects that occur contemporaneously in space and time. Thus, as described in Section 2.5.7, an implementation of the MPF should represent invariant spatiotemporal features (i.e. sequences of features), which is the focus of this chapter.

As described in Section 6.4, the objective has been to expand the stationary time properties of the Hierarchical NBP (HNBP) algorithm to facilitate sequential time-series processing. In the following we describe the most relevant components of the HNBP to address when expanding to facilitate sequential time-series processing.
In Section 6.2.1 we described an important property of the HNBP hierarchical network to be the compression of space as the hierarchy is ascended. We described that, as parent nodes “accumulate” the scope of their children, they “overlook” larger and larger spatial areas. In this chapter, we expand the HNBP framework into the time dimension. As a consequence, parent nodes of the DHNBP framework overlook both broader spatial areas but also longer time sequences of features compared to their child nodes. So parent nodes both span the spatial and temporal scope of their children. As described later in Section 7.2.2, a consequence of this is that as we ascend the hierarchy, we both have a compression of space and time. The compression of time means that nodes in higher layers of spatial scale exist in coarser time resolutions. We refer to this concept as Slow Feature Analysis, which we will describe later in Section 7.2.2.

In Slow Feature Analysis, all features on the same level exist in the same resolution of time. However, different features on the same layer will still change state faster or slower than other features on the same layer and thus do not necessarily evolve identically in time. We call this concept variable-duration state modeling. The concept of variable-duration modeling adds another layer of complexity to how spatiotemporal features are represented.

The last time-relevant component of the HNBP framework is the ability for parent nodes to “teach” child nodes. Information sent from parents, are sent to the Dirichlet Process (DP) for the specific child node. Thus, as the DHNBP framework extends the HNBP framework to facilitate time, it is necessary for the DHNBP to implement a time-varying DP that ensures that the properties of the DP are preserved in time-series analysis.

Our discussion so far has described three major components for the DHNBP framework to implement:

1. The extension of the stationary Dirichlet Process (DP) into the time dimension
2. Modeling different scales of temporal resolution (Slow Feature Analysis)
3. Modeling variable-duration sequences of features

In the following section (Section 7.2), we will describe the three main features of the DHNBP framework as listed immediately above.

7.2 DHNBP Features

In designing the DHNBP we have drawn on the experiences from a number of sources. The DHNBP model is essentially a Dynamic Markov Network model (Shen, Hengel, Dick & Brooks 2004, Hua & Wu 2007). However, based on the discussion of the preceding section, the DHNBP model has to incorporate three additional components (time-varying DP, hierarchical time compression and variable-duration sequences). Thus we build the DHNBP as a Dynamic Markov Network, but constructs a novel network and accompanying
set of equations for belief propagation. We start by addressing the time-series extension of the DP.

### 7.2.1 Time-varying Dirichlet Process

In our pursuit of learning invariant spatiotemporal features, we in the next proceeding sections expand our work with the stationary HNBP to model time. As described in Chapter 6, HNBP is able to form invariant spatial representations of objects. In this chapter we expand the HNBP to facilitate invariant spatiotemporal features. As described in the preceding section, it is first necessary to expand the stationary DP of the HNBP framework to time-series data.

In Section 6.2.3 the role of the DP in HNBP was described as to provide parents the ability to “teach” children. Thus, parents send children causal messages that are processed by the children’s DP. Each DP is then used to sharpen the beliefs of the children. In Section 4.7 we described how the stationary DP could be expanded to process time-series data. This work was inspired by Caron et al. (2007) that suggests introducing two additional steps to the CRP algorithm (Algorithm 4). This two step process was named DDP(). With the addition of the DDP process, the randomness of the DP is preserved for any time step of the Dynamic Markov Network. In this chapter, we suggest the utilization of the DDP process in the DHNBP framework to facilitate time-series causal message processing with the DP. We will further describe the role of the DDP in Section 7.4, but otherwise refer to Section 4.7 for more specific details of the DDP process.

### 7.2.2 Slow Feature Analysis

In designing an algorithm for representing invariant spatiotemporal features, we are again inspired by Dean (2006). Dean describes how the work by Wiskott & Sejnowski (2002) has inspired Dean’s DHHMM model. Wiskott & Sejnowski (2002) presents an algorithm for learning features from time-series data, called Slow Feature Analysis (SFA), based on work by Földiák (1991) on invariance. The object of SFA is to identify features present in data based on the assumption that the input data varies quickly while the features, we wish to learn varies slowly. Wiskott & Sejnowski (2002) expresses this as:

*The assumption is that primary sensory signals, which in general code for local properties, vary quickly while the perceived environment changes slowly. If one succeeds in extracting slow features from the quickly varying sensory signal, one is likely to obtain an invariant representation of the environment.*

Based on the above quote, we have therefore found it important to create a spatiotemporal hierarchy where as the hierarchy of nodes is ascended, sequences of feature representations change slower and slower. Thus, as mentioned in Section 7.1, the resulting DHNBP
framework will thus facilitate a compression of space and time as the hierarchy is ascended (Dean 2006).

In fig. 7.1 an illustration of the SFA concept can be seen. As the spatial hierarchy is ascended, features tend to vary slower and slower in time. In this specific example, the temporal dynamics of driving a car is being modeled. In the top layer, the driving environment of the car is being modeled. Below that layer, the speed of the car and below that the strain on the engine. We see that generally, the engine strain is high in the city environment and the speed is low and generally vice versa when traveling in a highway environment. Furthermore, we see that high-level features, as the driving environment, persist longer than low-level features as the engine strain and that the high-level driving environment is composed of many lower-level causes in time and space. The example of fig. 7.1 is an intuitive example of how real world events are coupled hierarchically in both space and time. It is both a nontrivial and important goal for implementations of MPFs to be able to represent such temporal hierarchies.

Based on our discussion so far, it is clear that the DHNBP model should implement a time compression (like SFA) as the hierarchy is ascended. One way to achieve this, is to force parent nodes to change state less frequently than their children by specifying an explicit longer waiting time (also known as sojourn time) between nodal states. Each node could be biased to stay in a state using a bias weight similar to Dean (2006). Another strategy is to simply let parents sample messages from children with a lower frequency than that of the children’s sampling frequency (Dean 2006). Thus, down sampling messages
as the hierarchy is ascended. We don’t further make any concrete decisions on how to implement the hierarchical compression of time, other than to describe its necessity and to provide the two ideas for implementation.

We now continue our description of the DHNBP features by expanding our discussion of waiting times between state transitions in Section 7.2.3.

7.2.3 Hidden Semi-Markov Model

As mentioned in Section 7.1, a main objective of the DHNBP model is the ability to represent variable-duration length sequences of features. In this author’s opinion, the ideal choice for modeling such a system is the Hidden Semi-Markov Model (HSMM) (Murphy 2002). In this section we introduce the main properties of the HSMM model that are relevant to the DHNBP framework.

Semi-Markov Process

The basis of the HSMM is a Hidden Markov Model (HMM) (Murphy 2001). With the basic HMM we are able to represent how a given hidden random variable, abstracting some measurable variable, changes over time. In fig. 7.2 an example of a hidden Markov model can be seen. Here, a measurable input variable $y$ is being abstracted by a hidden variable $x$ for three time steps, $t - 2$ to $t$. A feature of the HMM is that at each time step, the hidden variable $x$ will change state with probability one to another state or remain in the same state. This means that the time steps between each instance of the hidden variables in the hidden Markov process are constant. However, this is not necessarily ideal if the desire is to model variable-duration sequences of features. Instead we would rather model the sojourn (waiting) time between each state change explicitly, thus being able to model a different waiting time for each time step for each feature. Modeling the hidden Markov process \( \{x^t\} \) in fig. 7.2 as a semi-Markov process\(^1\) instead, means introducing

\(^1\)The term “semi” in “semi-Markov process” refers to the fact that the process can only be considered Markovian at the time when the processes changes state. In between state changes, the process is non-
an extra random variable (the remaining sojourn time $\tau$) that accounts for the variable sojourn time between hidden states. This effectively means, that the next state of the hidden variable $x^t$ is dependent not only on the previous state $x^{t-1}$ (as in fig. 7.2), but also the amount of time the process has stayed in the previous state as determined by $\tau^{t-1}$ (as illustrated in fig. 7.3). The sojourn time $\tau$ can be interpreted as a counter that can be decremented and once it reaches zero, $\tau$ triggers a change in state from (for instance) $x^{t-1}$ to $x^t$. Then, when moving to a new state $x^t$ from $x^{t-1}$, a new $\tau^t$ is determined that defines the new sojourn time. Thus, the new $\tau^t$ then determines for how long the process should stay in state $x^t$ before moving on to state $x^{t+1}$.

The object of DHNBP is essentially to perform Bayesian analysis across time, known as sequential Bayesian filtering. Sequential Bayesian filtering is often applied to simple HMMs (as in fig. 7.2), where the aim is to infer the state of the hidden variable sequentially (Murphy 2001). For instance, the state (or posterior distribution) of $x^t$ at time $t$ in fig. 7.2 can be inferred by applying Bayes’ theorem (see also eq. 4.1) as:

$$p(x^t|y^t) = \alpha p(y^t|x^t)p(x^t|y^{t-1})$$

(7.1)

where $\alpha$ is a normalizing constant that ensures that the posterior $p(x^t|y^t)$ is a proper probability distribution, $p(y^t|x^t)$ is the likelihood and $p(x^t|y^{t-1})$ is the prior distribution. When applying Bayes’ theorem to the semi-Markov model of fig. 7.3, the situation becomes Markovian because a change to a new state $x^t$ depends on both $x^{t-1}$ and $\tau^{t-1}$. We remind the reader that a Markov process, is a stochastic process that possesses the Markov property. Thus, in a Markov process, the probability of changing to a new state is only dependent on the present state of the process.
7.2 DHNBP Features

slightly more complex because of the sojourn ($\tau$) variable dependency:

$$p(x^t, \tau^t|y^t) = \alpha p(y^t|x^t)p(x^t, \tau^t|y^{t-1})$$

(7.2)

Thus, we now infer the state of both the $x$ and $\tau$ variable with each step of the semi-Markov model. We compute eq. 7.2 sequentially as determined by the sojourn times of each step. Thus when the sojourn time at, for instance, time $t-1$ has run out, we compute the next state for the semi-Markov chain at time $t$. This involves both computing the next state $x^t$ but also determining a new sojourn time $\tau^t$ by computing eq. 7.3. The next state of the semi-Markov process $x^t, \tau^t$ is computed as:

$$p(x^t, \tau^t) = \int \int p(x^{t-1}, \tau^{t-1})p(x^t|x^{t-1}, \tau^{t-1}), p(\tau^t|x^t, \tau^{t-1})dx^{t-1}d\tau^{t-1}$$

(7.3)

Referring to fig. 7.3, we see that the joint probability of $x^t$ and $\tau^t$ in eq. 7.3 is made up of three smaller distributions that we integrate over (with regards to to $x^{t-1}$ and $\tau^{t-1}$) to achieve the next state value. In this report, we recommend drawing samples from this integral using Monte Carlo integration via either the Gibbs or the importance sampler of Chapter 3 to keep computation tractable.

**Sequences of features**

So far in this section, we have described how to model the variable sojourn time between states, thus partly achieving the ability to model variable-duration sequences of features. We now conclude our application of the HSMM model (Murphy 2002) in the DHNBP framework by adding the ability to represent sequences of features.

In a hierarchical model (as the HNBP), nodes in higher layers model nodes in lower layers. When this property is combined with the hierarchical time compression of the SFA concept, the consequence is that each parent node abstracts a sequence of states (features) of each child node in a lower layer. In order to implement this property, the semi-Markov process of the HSMM is combined with the HNBP framework. Building on the notation of the previous section, we define the generalized state $G^t$ at time $t$ as:

$$G^t = (x^t, \tau^t)$$

(7.4)

As described by Murphy (2002), each generalized state of the HSMM can then generate a sequence of sub-states (features). We denote this sequence $z(G^t)$. Thus, the DHNBP is designed as a generative hierarchical model where parent nodes can generate sequences of children states (features). By combining the semi-Markov process with the notion of layers, we thus obtain a model where sequences of features can be generated. More formally, the generated sequence of states by node $x$ at time $t$ at layer $\ell$, $z(G^t_\ell)$, is the sequence of

2Note, that we have left out $\tau^t$ of the likelihood distribution $p(y^t|x^t)$ of eq. 7.2, because we in fig. 7.3 assume that $y^t$ is independent of $\tau^t$. 

---
Figure 7.4: Semi-Markov model with the ability to model sequences of features of lower spatiotemporal scale. The “virtual” node $z$ represents a sequence of features in a lower layer generated by the node $x$.

generalized states of a node $v$ in the layer below ($\{G_{\ell-1}^\ell\}$), as described in fig. 7.4 and eq. 7.5:

$$z(G_{\ell}^\ell) = \{G_{\ell-1}^{\ell}\}^\ell = \{(v_{\ell-1}^{\ell}, \tau_{\ell-1}^{\ell})\}^\ell$$  \hspace{1cm} (7.5)$$

Note that we distinguish between the time $t$ in layer $\ell$ and the time $\hat{t}$ in layer $\ell - 1$. As mentioned in the preceding section (Section 7.2.2), by implementing the concept of Slow Feature Analysis (SFA) in the DHNBP framework, we are able to model different time resolutions at different layers (resulting in time compression as the hierarchy is ascended). Thus, the resolution of time $t$ in layer $\ell$ is different from time $\hat{t}$ in layer $\ell - 1$. This means that all nodes in layer $\ell$ all exist in the same time resolution reference ($t$). Similarly, nodes in the layer below $\ell$ (i.e. $\ell - 1$) exist in their own time resolution reference ($\hat{t}$).

In summary, there are now two concepts related to modeling time: The first concept of SFA (Section 7.2.2) defines different resolutions of time at different layers of hierarchical network scale. The second (new) concept models sequences of features in a hierarchy. By combining the two features, the ability to model variable-duration sequences of features in a hierarchy of spatiotemporal scale is obtained.
7.2.4 Invariance in DHNBP

In Section 6.2.5 the stationary spatial invariant properties of the HNBP framework was described. In this chapter, the stationary HNBP framework was extended to process time-series data, resulting in the DHNBP framework, which is able to facilitate invariant spatiotemporal feature representation. The invariance is a property of the DHNBP based on its parts:

- **HNBP**
  HNBP allows for arbitrary topologies, thus facilitating overlapping receptive fields and data redundancy. As mentioned in Section 6.2.5, overlapping fields and redundancy is an important part of achieving invariance (Ullman & Soloviev 1999) and "explaining away".

- **Time-varying DP**
  The time-varying DP (Section 4.7) clusters time-varying causal feedback messages. This means clustering and interpreting similar data into discrete categorial representations which also leads to invariant representations.

- **Message Passing**
  Message passing is done across spatial hierarchies and beliefs are computed across time (further described in Section 7.4.2), allowing for invariant representations to be propagated, thus leading to (more) invariant representations.

- **Slow Feature Analysis**
  The DHNBP implements a time compression as the hierarchy is ascended. As described by Wiskott & Sejnowski (2002), by capturing slow features from the quickly varying sensory environment, invariant representations can be formed. Furthermore, because the sequence of features is modeled as a semi-Markov processes, the DHNBP model acts as a smoothing filter (Dean 2006), which also aids the process of slow feature analysis and thus facilitating invariant representations.

By combining these properties of the DHNBP we obtain the ability to represent invariant spatiotemporal features.

7.3 Overview

Before describing the DHNBP algorithm, we briefly provide an overview of the DHNBP model. The model in fig. 7.5 graphically summarizes the features of the novel DHNBP model. With the DHNBP model we have achieved:

- Application of the novel HBNP framework, yielding spatial compression as the hierarchy is ascended.
Figure 7.5: Illustration of DHNBP model. The illustration shows that each DHNBP node (white) has three associated nodes: Time-varying DP node (black), sojourn variable $\tau$ (red) and input node (gray, optional). The illustration also shows the dynamic aspect of the model as the stationary HNBP network is extended into the time dimension. In this illustration we have shown the dependencies of the Markov model when moving from time step $t - 1$ to $t$. We see that each state in time is modeled as a semi-Markov process. This provides the ability of the DHNBP model to model sequences of features of lower spatiotemporal scale.

- Application of the Slow Feature Analysis concept, yielding temporal compression as the hierarchy is ascended.
- Application of the HSMM framework (Murphy 2002), yielding variable-duration sequence of features modeling, extending the stationary HNBP into time.
- Temporal and spatial compression together facilitates spatiotemporal feature representation.
- Application of time-varying DP to allow parent nodes to “teach” children nodes.
- Invariance facilitated by smoothing property of the temporal HSMM, clustering property of DP and inference and prediction in HNBP message passing.
- Input nodes mandatory for DHNBP nodes on lowest layer (input layer), input optional for higher layer nodes.
- DHNBP belief propagation algorithm based on simulation and nonparametrics (will be described next in Section 7.4).
In the next section, we describe the overall algorithm and equations that have been developed for the DHNBP framework.

7.4 DHNBP Algorithm

In this section we describe the DHNBP algorithm, which is an extension of the stationary HNBP algorithm into the time domain. Thus, the DHNBP is identical to the HNBP algorithm of Section 6.3, but with a temporal dimension.

The DHNBP algorithm is built on HNBP Algorithms 7 and 8. Thus in DHNBP, Phase One (Algorithm 7) of the HNBP algorithm is performed until beliefs have settled. Hereafter, Phase Two (Algorithm 8) is executed, where beliefs of nodes are computed and the DPs for each node are “overwritten”. However, in DHNBP we add that Phase One and Phase Two are performed in between time steps. Therefore, after moving to a new time step and before executing Phase One and Two again, it is necessary to initialize the time-varying DP based on the DDP process (see Section 7.2.1). Hereafter Phase One and Two can be computed. The overall DHNBP algorithm is thus:

1. At time $t - 1$:
   (a) Initialize time-varying DP based on the DP from time $t - 2$
   (b) Perform Phase One and Two of HNBP algorithm (i.e. compute beliefs)
   (c) When time runs out as determined by $\tau^{t-1}$: move to time $t$

2. At time $t$:
   (a) Initialize time-varying DP based on the DP from time $t - 1$
   (b) Perform Phase One and Two of HNBP algorithm (i.e. compute beliefs)
   (c) When time runs out as determined by $\tau^t$: move to time $t + 1$

3. At time $t + 1$: etc...

In the following we continue to describe the equations of the DHNBP algorithm at time step $t$ when moving from time step $t - 1$.

7.4.1 Phase One: DHNBP Message Construction

When moving to a new state from time $t - 1$ to $t$, the prior (DP) from previous time step has to be updated using the two step updating policy by Caron et al. (2007), denoted by $\text{DDP}(\cdot)$. Thus the new DP for node $x_k$ at time step $t$ is based on the DP from the previous time step $t - 1$:

$$
\gamma^t_k(x^t_k) \leftarrow \text{DDP}(\gamma^{t-1}_k(x^{t-1}_k)) \quad (7.6)
$$
7.4 DHNBP Algorithm

Hereafter message passing and construction can proceed as described in the HNBP Phase One algorithm (Algorithm 7): first, the object is to update the DP $\gamma^t_k$ with samples from new causal messages. We define the causal message product at time $t$ as:

$$\Psi^t_{pa(x^t_k)}(x^t_k) \triangleq \prod_{\forall n \in pa(x^t_k)} m^t_{n,k}(x^t_k)$$ (7.7)

where $pa(x^t_k)$ denotes the parent nodes of node $x_k$ at time $t$. Hereafter, $M$ samples ($\{S_n\}_M$) are drawn from the causal message product using the Gibbs sampler of Algorithm 5:

$$\{S_n\}_{n=0}^M \sim \Psi_{pa(x^t_k)}(x^t_k)$$ (7.8)

Finally, we update the DP $\gamma^t_k$ at time $t$ with the new samples $\{S_n\}_M$ using the CRP algorithm of Algorithm 4:

$$\gamma^t_k(x^t_k) \leftarrow \text{CRP}(\gamma^t_k(x^t_k), \{S_n\})$$ (7.9)

After receiving causal evidence, node $x_k$ can send a message to node $x_j$ at time step $t$, based on the incoming neighborhood messages and the local evidence:

$$m^t_{k,j}(x^t_j) \leftarrow \alpha \int_{x^t_k} \phi^t_k(x^t_k) \gamma^t_k(x^t_k) \psi^t_k(x^t_k, x^t_j) \prod_{\forall n \in ne(x^t_k) \setminus j} m^t_{n,k}(x^t_k) dx^t_k$$ (7.10)

Eq. 7.10 is computed using Algorithm 7 of HNBP. So far, the DHNBP algorithm is similar to the HNBP algorithm only with the addition of the initializing DDP step and time indices. Phase One for time step $t$ continues (as in the HNBP Phase One algorithm) until the system is believed to have reached a steady state. Hereafter the DHNBP can enter Phase Two.

7.4.2 Phase Two: DHNBP Belief Update

Phase Two of the DHNBP algorithm is similar to its counterpart in HNBP (Algorithm 8), however the computation of the belief of node $x_k$ is based on the belief from the previous step: $b^t_k(x^t_k|x^{t-1}_k)$. So in contrast to the the HNBP algorithm, the belief of a DHNBP node is conditional on the belief of the last time step. Furthermore, as mentioned in Section 7.2.3, we model each DHNBP node as a variable-duration semi-Markov process in the DHNBP algorithm. As shown in eq. 7.2, the posterior distribution therefore possesses slightly more complexity than otherwise because of the sojourn dependency. We further showed that the next state (or belief) of the semi-Markov process could be computed as in eq. 7.3. By combining the belief computation from HNBP (eq. 6.5) with eq. 7.3, we

---

3The DHNBP network is a first order Markov network in the time dimension, thus it is only necessary to include the belief of the last step when computing the belief of the current step.
extend the HNBP network into time. Thus, the belief of node $x_k$ at time $t$, based on the previous time step $t - 1$, can be computed as:

$$b^t_k(x^t_k|x^{t-1}_k) \leftarrow \alpha \phi^t_k(x^t_k) \gamma^t_k(x^t_k) \int p(x^t_k, \tau^t_k) d\tau^t_k \prod_{\forall n \in \text{ne}(x^t_k)} m^t_{n,k}(x^t_n), \quad (7.11)$$

In eq. 7.11 we integrate over eq. 7.3 with regards to $\tau^t_k$, to get the next state of the $x^t_k$ variable in the semi-Markov chain. Notice that eq. 7.11 is similar to its counterpart in HNBP (eq. 6.5) only with the addition of the sojourn time dependency (described in Section 7.2.3).

As with HNBP, in the last step of Phase Two, the DP at time $t$ is “overwritten” with the newly computed belief:

$$\gamma^t_k(x^t_k) = \text{CRP}(b^t_k(x^t_k|x^{t-1}_k)) \quad (7.12)$$

Again, in eq. 7.12 we convert the belief distribution to a proper DP distribution using the CRP(·) function. Note, Phase Two of DHNBP is mandatory before moving on the next time step and starting over with Phase One.

We do not provide a pseudo-code algorithm for DHNBP Phase One and Two (updated versions of the HNBP algorithms (Algorithms 7 and 8) as in Section 6.3.3). This is mainly because we have not been able to determine a specific sampling algorithm for drawing samples from the next state distribution (eq. 7.3) in eq. 7.11, within the time constraints of this master thesis project.

### 7.5 Sub-conclusion

In this chapter we have described the Dynamic Hierarchical Nonparametric Belief Propagation (DHNBP) framework. The framework is an extension of the Hierarchical NBP (HNBP) algorithm of Chapter 6 to facilitate time-series processing.

The DHNBP algorithm essentially performs so-called sequential Bayesian filtering, thus applying Bayes’ theorem (eq. 4.1) to a spatiotemporal hierarchy of nodes. In Section 7.1, it was described that when extending the stationary HNBP framework towards time-series processing, three issues had to be addressed. In Section 7.2 we described how these issues are resolved in the DHNBP framework:

1. In Section 7.2.1 it was described how the stationary DP, associated with each HNBP node had to be extended to handle time-series processing. This role was fulfilled by the time-varying Dirichlet process (originally described in Section 4.7) in DHNBP.

2. In Section 7.2.2 the application of the concept of Slow Feature Analysis and its purpose in facilitating invariant representations in the spatiotemporal context of the DHNBP was described.
3. In *Section 7.2.3* the application of the semi-Markov model to DHNBP, to model variable-duration sequences of features, was described.

The description of the DHNBP framework was concluded in *Section 7.4* by outlaying the main components of the DHNBP algorithm.

The ability to model objects in space and time was described in *Section 2.5.7* as an important part of an MPF. Thus, by implementing invariant spatiotemporal representation of sequences of features in space and time, the DHNBP algorithm has implemented an important goal. Inference and prediction is performed using computationally tractable methods based on the approximative HNBP algorithm which again is founded on the Non-parametric Belief Propagation algorithm by Sudderth (2006). Furthermore, the DHNBP algorithm was designed to allow parent nodes to “teach” their children nodes by allowing causal feedback messages to shape the beliefs of their children using the time-varying DP. Time compression is implemented via the concept of Slow Feature Analysis by allowing nodes in different layers of spatial scale to evolve in different references of time. We furthermore, applied the use of the semi-Markov chain to model variable-duration sequences of features. Featuring variable-duration modeling is important to represent how different sequences of features evolve differently over time.

In conclusion, we have derived the novel DHNBP framework by extending the stationary HNBP network into time. However, we note that Phase Two of *Section 7.4.2* still has open-ended aspects. Unfortunately, due to the time constraint of this master thesis, it was not possible to derive a specific sampling algorithm for *eq. 7.11* in order to draw samples from the next state distribution of *eq. 7.3*. We did however, provide an overall algorithm.
Chapter 8

Conclusion

In this chapter we provide the main conclusion of the report. First, we describe the overall achievements of the master thesis project and then conclude the chapter by describing recommended areas of future research.

8.1 Conclusion

The objective of the master thesis project has been to describe a model for implementing the Memory-prediction Framework (MPF) by Hawkins & Blakeslee (2004). The model has been named the Dynamic Hierarchical Nonparametric Belief Propagation (DHNBP) framework and can be considered one of many possible solutions to implementing the concepts of the MPF. As described in Section 2.4.1, the Hierarchical Temporal Memory (HTM) implementation of the MPF by George & Hawkins (2005) has so far, in this author’s opinion, not been able to implement some of the most fundamental concepts of the MPF (such as the support for temporal analysis). Thus, the DHNBP framework aims to fully natively implement the concepts of the MPF.

A cornerstone of the HTM implementation is the Belief Propagation algorithm by Pearl (1988). The assumption of this project has been that using exact computation schemes, as the Belief Propagation algorithm, leads to intractable computation when the algorithm is applied to complex problems (as described in Section 2.2.3). Instead the theme has been that in order to create a computationally tractable solution that scales well with complexity, it is necessary to depart with the ways of exact computation. In this project we have been inspired by Lee & Mumford (2003) in designing a computationally tractable solution (i.e. the DHNBP framework). As mentioned in Section 2.5, Lee & Mumford (2003) suggests using particle filtering as the basis for the reasoning scheme of the probabilistic network instead of using the Belief Propagation algorithm. The general theme of the project has thus been to approximate solutions using particle filters and Markov Chain Monte Carlo techniques. The result is a novel theoretical framework for implementing the concepts of the MPF only using computationally tractable techniques.
In Section 2.5.7 we described a series of overall goals for the alternative solution (i.e. the DHNBP framework) to implement. We now in the following conclude on each of those categories:

- **Belief Propagation**
  Belief propagation has been implemented as a particle filter. The belief propagation algorithm scheme passes beliefs between nodes of different spatiotemporal scales using MCMC techniques. The DHNBP algorithm is based on the Nonparametric Belief Propagation (NBP) algorithm by Sudderth (2006). Thus, we have extended the hierarchically flat NBP algorithm into a framework where beliefs are passed between nodes of different spatial context (implemented as the HNBP algorithm of Chapter 6). Furthermore, the HNBP algorithm has been extended into the time domain (DHNBP algorithm of Chapter 7) to facilitate temporal processing. We have further described how the time-varying Dirichlet process (DP) is incorporated into the DHNBP framework. The time-varying DP allows parent nodes to become “teachers” for child nodes. This is done by using causal messages from parent nodes to update the DP associated with each child node. The DP was found to be especially well-suited for the DHNBP framework given its nonparametric nature.

Both the HNBP and DHNBP algorithms are of our own invention (based on NBP) and are novel contributions to the research field of combining computationally tractable nonparametric methods with belief propagation in space and time.

- **Nonparametric Representation**
  A general theme of this thesis has been nonparametrics. As nonparametrics was an integral part of NBP it was natural to extend the use of nonparametrics to other areas of this thesis. Thus, the DHNBP framework is inherently nonparametric in that all distributions are modeled as convex combinations of smaller parametric parts. Furthermore, the HNBP algorithms (Algorithms 7 and 8) have been developed as particle filters, which is an ideal fit with the nonparametric strategy.

- **Invariance**
  As described in Section 7.2.4, the DHNBP framework is able to represent invariant spatiotemporal sequences of features. This is achieved by combining the clustering properties of the time-varying DP with the cross-hierarchical inference and prediction of the HNBP algorithm and the temporal hierarchical compression of the DHNBP network. Furthermore, we put no restrictions on the topology of the DHNBP network and support overlapping receptive fields which is an important part of achieving invariance (described in Section 7.2.2).

As mentioned in Section 2.5, the overall goal for this master thesis project has been to propose an alternative to the HTM model based on more computationally tractable
methods. Based on the conclusions of this section we overall conclude that we have been able to achieve the goals of Section 2.5 and has thus proposed an alternative theoretical model to the HTM. However, as we noted in Section 7.5, the DHNBP framework still has some open-ended aspects, that needs further research, before an implementation can be programmed. We described these aspects in the next section (Section 8.2).

We end the conclusion by noting that the DHNBP framework, proposed in this thesis is simply one specific solution that incorporates the concepts of the MPF. We have not however, investigated other variants of the DHNBP although we expect that many other variants could be defined that also incorporate the concepts of the MPF. The main contribution by this thesis is the research that lies behind discovering and investigating the components of the DHNBP that all fit into the computational tractability paradigm, which is a nontrivial and state of the art research area in modern artificial intelligence. Whether or not the DHNBP will in fact prove to scale well with complexity, when it is applied to real world problems, is something that cannot be concluded with certainty before actually implementing and testing the framework.

8.2 Future Research

As mentioned, this master thesis project is purely a theoretical contribution to the research in modern artificial intelligence and it has not been the focus of this project to implement the developed theories. However, we feel that the next step, after having developed the theories of the DHNBP, is to implement them and learn from the implementation. The developed HNBP algorithm is sufficiently detailed to allow for direct implementation, however, the DHNBP framework still has unresolved aspects, as we noted in Section 7.5. We describe these aspects further in the following.

8.2.1 DHNBP Algorithm

In Section 6.3.3 we described the HNBP algorithm to a detailed degree. Unfortunately, we do not offer a similar detailed description of the DHNBP algorithm in Section 7.4. This is largely because the DHNBP algorithm has aspects that we were not able to solve within the time constraints of this master thesis project. The following aspects of the DHNBP algorithm remain open-ended:

1. In Section 7.2.2 the importance of time compression, as the hierarchy is ascended, was described. Two ideas for implementation were offered, but no further implementation specifics were given.

2. In Section 7.2.3 it was described, that whenever the semi-Markov process changes state, a new hidden variable state ($x$) and sojourn time ($\tau$) is sampled from eq. 7.3. However, we did not provide any specifics regarding how such a sampling could be performed or how the distributions of eq. 7.3 of Section 7.2.3 could be learned.
3. Lastly, in eq. 7.11 of Section 7.4.2 it was shown how the belief of a node in time step $t$ is dependent on the state and sojourn time of the previous time step $t-1$. However, we did not offer a specific sampling algorithm for determining this equation.

We believe that these subjects are the most relevant and immediate open-ended problems to solve before implementing the DHNBP framework.

### 8.2.2 Learning

We have not described any algorithms for learning in neither the HNBP nor DHNBP framework. The focus of this thesis has solely been on discovering tools and methods for representing features. However, in order to carry out an implementation of the MPF it is necessary to facilitate both representation and learning of invariant spatiotemporal sequences of features. The development of learning algorithms would allow the model to learn distributions autonomously and unsupervised. The distributions that need to be learned are the potential functions between the nodes of the DHNBP framework and the distributions of the semi-Markov chain. In Detry & Piater (2007), potential functions are created automatically by simply subtracting samples from each of the attached nodes’ belief distributions, thus letting the potential function represent a distance in euclidean space. The solution by Detry & Piater (2007) is attractive and intuitive and fully unsupervised. In other literature (Sudderth et al. 2003, Isard 2003), we have found that a simple solution is to manually provide suitable distributions. Often, the system designer can create reasonable guesses for what these distributions and potentials should be, but such guesses are only appropriate for development purposes and are not useful as permanent solutions. We note that learning is an integral part of the unsupervised MPF and should also be the focus of further research.

### 8.2.3 Application & Implementation

Once the remaining aspects of the DHNBP framework have been addressed and learning algorithms have been developed, the system can be applied and tested. As we noted in Section 2.5.6, video is a type of source that the DHNBP (and MPFs in general) are able to model very well because of their inherent spatiotemporal nature. However, when testing an application, video data could prove cumbersome and we therefore suggest testing an application to still images first. The domain of computer vision is not the only relevant area of implementation though. A DHNBP implementation should be able to learn and represent features present in any structured data. In fact, if the DHNBP module is implemented to be separate of the application (as we described in Section 2.1.1), the area of application is only specific to the pre- and processing modules. Thus, the DHNBP implementation itself should be a generic module that ideally is unaffected by the choice of application. Therefore, applying the system should amount to developing suitable pre-
and post-processing modules. As we mentioned in Section 2.5.6, when dealing with computer vision, especially ASSOMs and SIFT-descriptors have been found interesting, and we therefore suggest considering them as preprocessing modules if the DHNBP should be applied to the domain of computer vision.
Appendix A

Approximation in Bayesian Networks

In this chapter, in-depth background on approximation in Bayesian networks is provided. Section A.1 describes how the posterior expectation can be approximated with Monte Carlo integration (a basis for doing MCMC in Bayesian networks). Section A.2 is an example of MCMC applied to a practical example of Gibbs sampling in a Bayesian network.

Note, this appendix chapter was originally written for the FORK project and published in Schwartz (2008b). We republish the chapter in this report as a service to the reader.

A.1 Integrals in Bayesian inference

The purpose of Bayesian inference is to compute posterior probability distributions. Let $D$ denote observed data, and $\theta$ denote the parameters of a model describing the data (a hypothesis). The joint probability, $P(D, \theta)$, can be computed as the product of the prior probability $P(\theta)$ and the likelihood $P(D|\theta)$:

$$P(D, \theta) = P(D|\theta) P(\theta) \quad (A.1)$$

In the context of eq. A.1, inference is performed by applying Bayes’ theorem to $P(D|\theta)$, which yields:

$$P(\theta|D) = \frac{P(\theta) P(D|\theta)}{\int P(\theta) P(D|\theta) d\theta} \quad (A.2)$$

Eq. A.2 is the posterior probability distribution of $\theta$. How the posterior distribution is used further on, is a matter of how the Bayesian network is applied. However, frequently it is desirable to compute the expected value of a random variable (r.v.) (computing the expectation of an r.v. is the same as predicting the value of an r.v.). The posterior expectation of some function $f(\theta)$ is then:
\[
E[f(\theta) | D] = \frac{\int f(\theta) P(\theta) P(D | \theta) d\theta}{\int P(\theta) P(D | \theta) d\theta}
\]  

(A.3)

Analytically determining the integrals of eq. A.3 can be problematic when dealing with high-dimensional integrals. However, alternatively we can use Monte Carlo integration (see also Section 3.2) which stochastically approximates eq. A.3 by drawing a large number \(N\) of samples \(\{\theta_t, t = 1, \ldots, N\}\). The expectation value can then be approximated as the mean of the samples:

\[
E[f(\theta) | D] \simeq \frac{1}{N} \sum_{t=1}^{N} f(\theta_t) | D
\]  

(A.4)

**A.2 Gibbs Sampling**

The purpose of this section is to provide further background on the Gibbs sampling algorithm (see also Section 3.6) and give a concrete example of the sampling process. Note that in the following sections, the terms conditional probability distribution and joint probability distribution have been abbreviated to CPD and JPD respectively.

**A.2.1 Application**

To illustrate the process of Gibbs sampling in practice please consider the belief network in fig. A.1. The belief network has four nodes: Sprinkler (S) and Wet Grass (W) are evidence nodes, and Rain (R) and Cloudy (C) are hidden nodes. CPDs are defined as, \(P(S|C)\), \(P(R|C)\), \(P(W|R,S)\). The task is to compute the posterior distribution of: \(P(R|W,S)\). This means that we will be performing inference by using the evidence provided by \(W\) and \(S\). Given the small network of the example this can easily be done exactly without using Gibbs sampling. However, for the sake of learning we will be applying the Gibbs sampling algorithm in order to compute the posterior distribution, by utilizing the CPDs in the Markov blanket of either \(C\) or \(R\). We are going to compute the JPD, \(P(C, R, S, W)\) and then \(P(R|W,S)\).

The first step in Algorithm 3 of Section 3.6.3 is to initialize all variables in the Markov chain. In our case \(R\) and \(C\) only takes on binary values so they are initialized by sampling from a binary uniform r.v. Hereafter, as in algorithm 3 we use a deterministic updating schedule where we start by updating \(C\) and then use the value of \(C\) to update \(R\).

**A.2.2 Example**

The following is a run-through of the Gibbs sampler applied to the network in fig. A.1. Each variable is conditioned on the other. After one variable is updated at step \(\tau\) the state of the variable is inserted into Markov chain \(X\).
1. $\tau = 0$

   (a) Initialize: Sample binary uniform r.v. $B$ twice. Say we get:
   
   i. $B^0_C = \text{true}$
   
   ii. $B^0_R = \text{true}$

   (b) Insert into Markov chain:
   
   i. $X^{(0)} = (C^{(0)} = \text{true}, R^{(0)} = \text{true})$

2. $\tau = 1$

   (a) Update $C$ (conditioning on $S, R, W$):
   
   i. Computing $P(C|S, R)$

   $$P(C|S = T, R = T, W = T) = P(C|S = T, R = T) \quad (A.5)$$

   We can leave out $W$ because it is not a part of $C$’s Markov blanket. The problem here is that we don’t have values for $P(C|S = T, R = T)$. However
A.2 Gibbs Sampling

if we apply Bayes’ theorem we can obtain probabilities we do have (in the CPDs already given). Applying Bayes’ theorem is in fact inference and is what we want to achieve:

\[
P(C|S,R) = \frac{P(C)P(S,R|C)}{P(S,R)} \tag{A.6}
\]

\[
= \frac{P(C)P(S|C)P(R|C)}{P(S,R)} \tag{A.7}
\]

All the probabilities in eq. A.6 are known except for the denominator, \(P(S,R)\). This probability can be considered a normalizing value that makes sure that the sum of all possible values of \(P(C|R,S)\) sums to unity. Given the fact that \(C\) is a binary variable it can only take on two values: \(C = T\) or \(C = F\). To evaluate eq. A.6 we need to write out the denominator to a sum of all of the probabilities that \(P(C|S,R)\) can take on.

For the sake of brevity I denote \(C = T\) by \(\wedge C\) and \(C = F\) by \(\lnot C\):

\[
P(C|R,S) = \frac{P(C)P(S|C)P(R|C)}{P(C)P(S|C)P(R|C) + P(\lnot C)P(S|\lnot C)P(R|\lnot C)} \tag{A.8}
\]

\[
= \frac{0.5 \times 0.1 \times 0.8 + 0.5 \times 0.5 \times 0.2}{0.5 \times 0.1 \times 0.8 + 0.5 \times 0.5 \times 0.2} \tag{A.9}
\]

\[
= 0.44 \tag{A.10}
\]

Note that in the evaluation of eq. A.8, \(R\) takes on the value it has from the last state in the Markov chain: \(R^{(0)} = \text{true}\), which in this iteration stems from the initialization at \(\tau = 0\). Of course \(S = \text{true}\), given that it is fixed.

ii. Sample from real uniform(0,1) r.v. U. Say we get:

\[U = 0.31\]

iii. If \(U > P(C = T|R,S)\) then insert \(C = T\) into \(X\), else \(C = F\). Given that \(U = 0.31\) then \(C = F\).

iv. Insert into Markov chain \(X\) (\(C\) is updated and \(R\) keeps it’s value from last step):

\[X^{(1)} = (C^{(1)} = \text{false}, R^{(0)} = \text{true})\]

(b) Update \(R\) (conditioning on \(C,S,W\)):

i. Computing \(P(R|C,S,W)\):

Using Bayes’ theorem we obtain:

\[
P(R|C,S,W) = \frac{P(R|C,S)P(W|R,C,S)}{P(W|C,S)} \tag{A.11}
\]

\[
= \frac{P(R|C)P(W|S,R)}{P(W|C,S)} \tag{A.12}
\]

Here we utilize the fact that \(R\) is conditionally independent of \(S\) given \(C\), and \(W\) is conditionally independent of \(C\) given \(S,R\). Again by using the
notation from earlier, we have:

\[ P(R|C, S, W) = \frac{P(R|C)P(W|S, R)}{P(R|C)P(W|S, R) + P(R\neg C)P(W|S, R)} \quad (A.13) \]

\[ P(R|C, S, W) = \frac{0.2 \times 0.99}{0.2 \times 0.99 + 0.8 \times 0.90} \quad (A.14) \]

\[ P(R|C, S, W) = 0.22 \quad (A.15) \]

\[ P(R|C, S, W) = 0.22 \quad (A.16) \]

We want to compute the probability of \( R = T \) and given the state of \( C \) in \( X^{(1)} \) being \( C^{(1)} = F \) we have the equation:

\[ P(R|C, S, W) = \frac{0.2 \times 0.99}{0.2 \times 0.99 + 0.8 \times 0.90} \]

\[ P(R|C, S, W) = 0.22 \]

\[ ii. \text{ Sample from real uniform}(0,1) \text{ r.v. U. Say we get:} \]

\[ U = 0.57 \]

\[ iii. \text{ If } U > P(R = T|C, S, W) \text{ then insert } R = T \text{ into } X, \text{ else } R = F. \text{ Given } U = 0.57 \text{ then } R = T. \]

\[ iv. \text{ Insert into Markov chain (R is updated and C keeps it’s value):} \]

\[ X^{(1)} = (C^{(1)} = \text{false}, R^{(1)} = \text{true}) \]

(c) \( \tau = 2 \)

\[ \vdots \]

The sampling continues for as many iterations desired.

Using a python implementation\(^1\) of the above Gibbs algorithm, 1,000,000 samples was drawn with evidence: \( S = T, W = T \). The resulting distributions was:

\[ P(R = T|.) = 0.320 \]

\[ P(R = F|.) = 0.680 \]

\[ P(C = T|.) = 0.175 \]

\[ P(C = F|.) = 0.825 \]

Hereafter, a second test was run. The evidence was changed to: \( S = F, W = T \). Meaning that the sprinkler is now off but the grass is wet. Again 1,000,000 samples were drawn. The resulting distributions was:

\[ P(R = T|.) = 0.992 \]

\[ P(R = F|.) = 0.008 \]

\[ P(C = T|.) = 0.873 \]

\[ P(C = F|.) = 0.127 \]

\(^1\)Originally developed by the University of Minnesota. Source code available on accompanying CD or can be downloaded from [http://www-users.itlabs.umn.edu/classes/Spring-2008/csci5512/index.php?page=resources](http://www-users.itlabs.umn.edu/classes/Spring-2008/csci5512/index.php?page=resources)
A.2.3 Interpretation of the results

Interpreting the results, it is intuitively likely that the Gibbs sampler successfully has approximated the JPD of the variables in the Markov blanket of both C and R.

For instance, it was seen that the result of $P(R = T|S = T, W = T) = 0.320$. This low probability of that it is raining makes sense given that we are measuring that the grass is wet, and that the sprinkler is on. That the sprinkler is on makes it (more) unlikely that rain is causing the grass to be wet. The second test run showed that when the evidence changes: $S = F, W = T$, the resulting probability: $P(R = T|S = F, W = T) = 0.992$ also shows very intuitively that given that the sprinkler is off, it must almost certainly be the rain that is causing the grass to be wet. This way of reasoning is known as “Explaining Away” (Pearl 1988).
Appendix B

Research Plan

In order to define the working relationship between the student and Sigma Space Corp., a research plan for both the FORK and master thesis project was defined (Section B.1). It aimed to ensure that efforts of the student, were as much as possible in line with goals that Sigma Space Corp. wished to pursue. In Section B.2, the exploration of the FORK project is discussed, showing the chronological exploration path of the research topics of the project. A similar description of the master thesis project can be found in Section B.3.

B.1 Research Plan

The research plan is divided into two phases. One for the FORK project and one for the future master thesis itself. The plan consists of topics that are relevant for developing an MPF.

1. FORK

   (a) Relate belief propagation in belief networks to biologically plausible neural nets (e.g. the artificial Hopfield net, Boltzmann machines) to show biological plausibility.

   (b) Understand and make the transition from exact, parametric belief propagation to approximate, nonparametric belief propagation.

   (c) Understand and research Nonparametric Belief Propagation (NBP) and understand how NBP is a generalized particle filter. Furthermore, understand the strengths and weaknesses of NBP, in order to identify problem areas (in relation as a solution to an MPF) that need attention in order to be implemented in the master thesis project.

   (d) Understand how a hierarchical NBP with Gibbs sampling provides feedback and resolves ambiguity.
2. Master Thesis

(a) Understand how the multiple MC iteration streams realize sequential Bayesian filtering (particle filtering) and allow data-driven clustering via random walks.

(b) Research and develop solutions to deal with the insufficiencies of NBP (i.e. support for vertical hierarchy and temporal sequence processing).

(c) Research NBP with MCMC Gibbs sampling in a sequential Bayesian filtering framework (thus amenable to general graphs, including arbitrary belief nets) that naturally iterates over priors, likelihoods, and posteriors, even hierarchically.

(d) Understand, experiment, speculate, and propose ideas about the above – being able to deal with general graph forms and structures – just how nonparametric models (where the data speaks for itself, thus allowing the number of clusters during training) can be used in a self-organizing and self-calibrating way.

B.2 FORK Project

Fig. B.1 illustrates the explored subjects of the FORK project. Please note that the graph is only partially true to the hierarchical relationships between the subjects, but first-most shows the chronological order of exploration of topics in the FORK project. The yellow path dictates the primary path of exploration.

In line with Step 1b of the research plan (Section B.1), the figure shows that the initial focus of FORK was to identify how to transition from exact belief propagation in Bayesian networks to approximation in Nonparametric Belief Propagation (NBP) in Markov random fields.

Initially the path of exploration digressed into self-organizing maps (SOMs) (see also Appendix D). However, after researching many different models of SOMs, it became apparent that no currently available SOMs would be able to provide a coherent solution to an MPF. However, this digression provided insight into the field of adaptive topology and “Discovery of structural form”. Especially Growing Neural Gas (GNG) and Instantaneous Topological Mapping Model (ITM) has proven interesting techniques for nonparametric adaptation of the topology of a graphical model.

After rejecting SOMs as solutions to MPFs, the focus was redirected towards Bayesian networks. It has been a theme from the beginning of the FORK project that exact belief propagation in Bayesian networks is computationally intractable and that approximation is necessary. This lead to further exploration of approximation methods in the domain of Markov Chain Monte Carlo (MCMC). Gibbs sampling, a popular MCMC method, is a vital component of NBP. Therefore, the strategy was to investigate Gibbs sampling in general before studying NBP, which proved beneficial. After thorough investigation of NBP it was concluded that NBP lacks the ability to model multiple layers of causes as
Figure B.1: Major topics explored in the FORK project. The yellow path of exploration denotes the most important topics researched. Note that the figure has a chronological hierarchy.

well as an explicit account for time and temporal sequence processing. This laid out the foundation for the master thesis project.
B.3 Master Thesis Project

Fig. B.2 illustrates the explored subjects of the master thesis project. The figure is a compromise between a chronological and hierarchical model. Thus, representation of the subjects are both partly true to their taxonomic composition and to the time line of the project. From fig. B.2 we also see that the project was generally approached using a top-down approach. Overall there are three major categories of research:

1. Prior Distribution
2. Belief Propagation
3. Sequential Bayesian Filtering

Unlike the more sequential exploration of the FORK project, the exploration of the master thesis project was inherently parallel. Thus, during the project, subjects within all three categories were researched at the same time. In the research area “Prior Distribution”, a suitable nonparametric prior distribution was researched. As described in Chapter 4, the Dirichlet Process was found to be a perfect fit for our needs. Especially the Time-varying Dirichlet Process Mixture Model by Caron et al. (2007) was found relevant.

In the research area “Belief Propagation”, the expansion of the Nonparametric Belief Propagation algorithm was researched. The resulting Hierarchical NBP (HNBP) algorithm was based on work from Isard (2003), Sudderth et al. (2003), Freeman & Pasztor (1999) and Detry & Piater (2007).

In the area of “Sequential Bayesian Filtering”, the coupling of the HNBP algorithm with the time domain was researched. Especially, the Dynamic Markov Network by Shen et al. (2004) was found interesting.

As mentioned before, all areas were researched in parallel. This was because all three areas had to fit with each other, so in order to make sure that the resulting solution would accommodate the desired features of Section 2.5.7, it was necessary to switch between the different research areas as new sources were read and ideas were developed. Thus, in the resulting model, the Dynamic Markov Network was coupled with the HNBP algorithm and the Time-varying DPM to produce the Dynamic HNBP (DHNBP) framework.
Figure B.2: Major topics explored in the master thesis project. The yellow path of exploration denotes the most important topics researched. Note that the figure has a chronological hierarchy.
Appendix C

Summary of Schwartz (2008a)

For the convenience of the reader, a short summary of Schwartz (2008a) is provided in this chapter to complement Section 2.4 of the main report. This paper was written by this author as the main part of the course AI03 at the University of Southern Denmark.

Note, this appendix chapter was originally written for the FORK project and published in Schwartz (2008b). We republish the chapter in this report as a service to the reader.

C.1 Introduction

Recently, there has been an increase in attention towards memory-prediction frameworks. In 2004, J. Hawkins published the book 'On Intelligence' (Hawkins & Blakeslee 2004). The book describes a coherent theory of how reasoning is done in the neocortex of the brain. According to Hawkins, the essential aspects of human intelligence occurs in the neocortex. The paradigm is essentially that by remapping the workings of the biological neocortex into a computational model, known as Hierarchical Temporal Memory (HTM), that can be implemented in computers, it is possible to achieve intelligent computing.

In 2005, Hawkins founded1 the research company Numenta. They focus their work on developing a system that will implement the features of the HTM as described in (Hawkins & Blakeslee 2004). Numenta have been releasing their work to the public under license but free of charge. The research release has been dubbed NuPIC (Numenta Platform for Intelligent Computing) and is a C++ computational engine with a Python interface.

The HTM model proposed by George and Hawkins (Hawkins & George 2007, George & Jaros 2007) is in essence a probabilistic graph model (a Bayesian network) and is based on the work done by J. Pearl in Pearl (1988).

As an alternative, Dean (2006) describes the Dean Hierarchical Hidden Markov Model (DHHMM). As well as Hawkins, Dean also bases his model on (dynamic) Bayesian networks but differs as he is able to solve fundamental issues, as the notion of time and

1Numenta was co-founded by Donna Dubinsky and Dileep George (www.numenta.com). George is one of the leading figures in the implementation of the NuPIC.
“explaining away”, which is not yet fully accounted for in the NuPIC implementation of the HTM model by Hawkins and George. Recently, Dean has worked with Google trying to implement his model but haven’t yet published any results from their combined efforts.

C.2 Conclusion

The HTM model and the DHHMM model have many similar features and basically try to achieve the same objectives but in different ways. However, it is clear that the NuPIC still lacks many of the fundamental concepts as originally described by Hawkins. It is the view of this author, that to some extent the DHHMM model is more true to the original HTM model, as that of the NuPIC implementation. The following list is a comparison of how both models handle three vital concepts of a memory-prediction framework:

- **Hierarchy**
  The DHHMM model is a polytree whereas the HTM model is a simple tree. A polytree enables the network to model a broader span of real-world objects. By designing the network as a simple tree structure the NuPIC implementation of the HTM model experiences computational simplifications but at the expense of expressiveness. One of the advantages of the polytree structure is that the network is able to facilitate “explaining away”, as described by Pearl in (Pearl 1988), which is not possible with a simple tree structure.

- **Time and Prediction**
  The HTM model does originally describe the notion of time during inference and prediction as a very important part of the HTM model as a synthetic model of the human brain. However, the NuPIC release currently does not fully implement time or prediction. The DHHMM model facilitates both time and prediction.

- **Memory**
  Both the HTM and DHHMM model suffer from memory state space explosion (Dean 2006, Garalevicius 2007) when trying to solve more realistic problems (i.e. using 4, 5 and 6 layers while having to learn large number of invariant features). It is the view of this author that the memory problem is the biggest problem facing memory prediction frameworks at the moment. Both Dean and Hawkins have performed their initial testing on “toy” problems (both using the “Pictionary” data set) and still publicly haven’t applied the systems to real-world problems.

  In conclusion, it is the view of this author that the limitations introduced in the design of the NuPIC is concerning to the viability of the HTM model as major limitations have already been placed before the system can be tested on real-world problems. The DHHMM model by Dean seems to provide many of the concepts that haven’t been implemented in the NuPIC implementation of the HTM model. But it is also worth noting
C.2 Conclusion

that the approach described by Dean, does not specify any information about how it can be implemented in practice. As a theory, the Dean DHHMM model seems to provide more flexibility than the HTM model as implemented by Numenta in the NuPIC, but as no (public) implementation has been presented by Dean so far, it is still uncertain if it will be possible to implement a practical model of the DHHMM model.
Appendix D

Alternative Approaches

D.1 Self-Organizing Maps

Self-Organizing Maps (SOMs) have been a focus of the FORK project by investigating whether SOMs, or any variants hereof, could fulfill the role of an MPF. The conclusion after some work was that no SOMs exists (to this author’s knowledge) that could be considered full solutions to MPFs and useable in neither the FORK nor the master thesis project. Furthermore, the conclusion has been that it would be too difficult to derive a SOM that could implement the requirements listed in Section 2.5.7. The impression of this author is, that there exists many excellent SOMs that perform very well, although often under specific simplifying assumptions, but there lack SOMs that combine the many different areas of an MPF, such as temporal sequence processing, statistical modeling of input data and hierarchical representation of high-order dependencies, into one unifying model.

In Section D.1.1, a brief introduction to SOMs are given, and a few of the most relevant SOMs in connection with MPFs are brought to the reader’s attention.

Note, this appendix chapter was originally written for the FORK project and published in Schwartz (2008b). We republish the chapter in this report as a service to the reader.

D.1.1 Introduction

SOMs were originally described in Kohonen (1982) and are unsupervised neural networks that specialize in creating maps that try to preserve the topological properties of the input data. They can take high-dimensional data and map them to lower dimensions which makes them ideal for visualization purposes. SOMs focus on grouping neurons together in order to be able to represent patterns of the input data. Traditionally, SOMs group neurons by analyzing the spatial arrangement of the input data in Euclidean space. However, some SOMs substitute Euclidean neighborhood functions with estimated posterior distributions by implementing Bayesian inference (BSOM: Yin & Allinson (1997), Yin &
D.1 Self-Organizing Maps

(Allinson (1998), Bayesian framework for SOM: Utsugi (1997)). This way SOMs can group neurons based on their statistical dependencies instead of Euclidean distances.

Often in SOMs, the topology of the network is fixed \textit{a priori} meaning that the learning of the input data is restricted to a specific topology given in advance. However, a very interesting area of SOMs is where the topology is learned from the data itself. By having a flexible network topology SOMs enjoy two added benefits: The first is, connections between nodes (that currently does not exist) can be created if the input data shows that associations in fact are present (CHL: Martinetz (1993)). Secondly, some SOMs also give the possibility to add new \textit{nodes} if the data speaks for it, and remove nodes that are no longer needed in order to be able to represent the data (Growing Neural Gas: Fritzke (1995), ITM: Jockusch & Ritter (1999), TreeGNG: Doherty, Adams & Davey (2005), Neural Gas for Sequences: Strickert & Hammer (2003)).

SOMs were not originally designed by Kohonen to account for the dimension of time. Therefore, several variants of the original SOM have been designed where temporal data is accounted for either by using recurrent (RSOM: Koskela, Varsta, Heikkonen & Kaskim (1997), Koskela, Varsta, Heikkonen & Kaski (1998)) or recursive feedback (RecSOM: Voegtlin (2002), Temporal Kohonen Map: Chappel & Taylor (1993)). For a comprehensive overview of time in SOMs, please refer to Barreto & Araújo (2000), Hammer, Micheli & Sperduti (2003) and Pölla (2005).

As mentioned, SOMs were initially a part of the focus for the FORK project. However, this author was not able to find any model that coherently implemented the goals described in \textit{Section 2.5.7}. Therefore, SOMs were abandoned early on in the research process. However, SOMs are attractive as they often, in an elegant and simple way, implement many of the goals in \textit{Section 2.5.7}, but very rarely all of them in \textit{one} model.

The SOM this author found most compelling was the HQSOM by Miller & Lommel (2006). HQSOM tries to implement both shift and scale invariant representation of objects by facilitating spatial and temporal sequence processing in a hierarchy of RSOMs. Miller & Lommel (2006) use a very similar argumentation for designing their algorithm as Hawkins & Blakeslee (2004) and Dean (2006). For an example, they all refer to work by Fukushima (1980), Mountcastle (1978) and Földiák (1991) on subjects as hierarchy, invariance and “explaining away”, to name a few. However, during the research phase it was unclear if HQSOM would scale up to real world problems given that the published test results in Miller & Lommel (2006) is based on “toy” problems. Furthermore, it was uncertain if the HQSOM would facilitate all the requirements listed in \textit{Section 2.5.7} based on the limited time available for research. Instead it was decided to research how to construct the MPF from the bottom up, instead of trying to coerce or expand the HQSOM, or any other SOM, into doing something it was not intended to do from the beginning.
D.1 Self-Organizing Maps

D.1.2 Adaptive Topology and the Discovery of Structural Form

As mentioned in Section 2.5.7, it is an attractive feature of an MPF to be able to deal with arbitrary network topologies and to do so in an automated and online fashion without having to fix or set any parameters in advance. In this report, this concept is known as Adaptive Topology (AT\textsuperscript{1}). In this context, “topology” refers to the arrangement of nodes in a graph. MPFs implementing such a strategy, would have data driven network topologies that adapt according to the input data, adding and removing nodes and node-connections on-the-fly as needed. An MPF with such a topology would enjoy maximum expressiveness and minimum computational efforts (as no unnecessary nodes or connections exists). However, this strategy raises the concern of how to deal with the flexibility and still have a computationally tractable system.

Different strategies exist in the literature. Some systems are fully automated and flexible (Fritzke 1995, Jockusch & Ritter 1999, Strickert & Hammer 2003). Such systems compute the topology from an infinite state space of topologies and do so whenever they detect it is necessary to recompute the topology. In other systems, the topology is selected from an a priori given (limited) set of topologies and have predefined intervals of recomputation (Kenp & Tenenbaum 2008). Many strategies exist in between these two extremes. However, in this report, only the most important in the literature are brought to the reader’s attention.

Growing Neural Gas (GNG)

A fundamental algorithm is the Growing Neural Gas (GNG) by Fritzke (1995). The GNG algorithm combines the algorithms Competitive Hebbian Learning (CHL) by Martinetz (1993) and Neural Gas (NG) by Martinetz & Schulten (1991). In essence, the algorithm iteratively measures input data $\xi$, distributed by some function $p(\xi)$, and tries to adapt a topology. This is done by measuring Euclidean distances between nodes in the topology and the current input data $\xi$. The existing node that is closest to $\xi$ is moved further towards $\xi$ along with that nodes’ neighbors. If the existing node that is second closest to $\xi$ has no connection to the closest node, a new connection between these two is made. If a connection already exists, the age of the connection is reset. Connections above a certain age are automatically removed. This way “patches” of the network move together and connections that are obsolete eventually disappear. Further, periodically new nodes are generated and inserted into the topology. The algorithm proceeds to adjust the topology and add new nodes until some stopping criterion is met (e.g. net size or performance measure).

To illustrate the capabilities of the GNG algorithm a Java applet developed by the University of Bochum, Germany\textsuperscript{2} was used. The applet is capable of simulating a number

\textsuperscript{1}AT is not a generally recognized term but is of this author’s own invention.

\textsuperscript{2}http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html
of different algorithms, including GNG, CHL and NG. In fig. D.1 the result from running the Java implementation of the GNG algorithm on can be seen. The dark gray area (shaped as UNIT) is the distribution of the input data. This means that the GNG algorithm should add nodes and connections that, as time progresses, adapt to this distribution. Fig. D.1.A shows how the GNG algorithm starts with two nodes. The position of the nodes are initially off as the GNG algorithm has no knowledge to begin with. As time progresses, the algorithm moves and adds nodes and connections as needed (fig. D.1.B). As more time progresses (fig. D.1.C), the GNG algorithm is able to approximate the distribution of the input data more precisely.

![Diagram](image.png)

Figure D.1: Screenshot from GNG Java applet developed by University of Bochum, Germany. In this example, when the GNG algorithm starts, only two nodes are used and are poor approximations to the distribution of the input data. As time progresses, more nodes and connections are added and the algorithm is able to approximate the distribution better. See text for more details.
Growing Neural Gas for Sequences

GNG is a powerful tool for forming network adaptive topologies. However, it fundamentally relies on the input data being temporally uncorrelated. When creating SOMs for use in, for instance, vision applications this is problematic. Video data, which in essence is a series of image frames, will often be closely correlated in time. This fact becomes apparent when imagining a camera panning over a scene. Frames recorded from this scene will most often be strongly correlated (i.e. one frame will look similar to the next). However, this strong correlation is problematic for the GNG algorithm (Jockusch & Ritter 1999).

The Instantaneous Topological Mapping Model (ITM) is a model that takes strongly correlated data into account (Jockusch & Ritter 1999). Thus, ITM is especially suited for online applications. Furthermore, according to Jockusch & Ritter (1999), the algorithm is computationally very efficient and is able to rapidly create topologies, increasing its suitability to online applications. Therefore, based on semantics of the GNG algorithm but with improvement for temporal sequences, the ITM algorithm is more suited for classic pattern recognition applications.

D.1.3 ASSOM

Adaptive-Subspace Self-Organizing Map (ASSOM) is a special type of SOM (Kohonen, Kaski, Lappalainen & Salojärvi 1997). The ASSOM specializes in its ability to learn transformation-, scale- and rotation-invariant filters unsupervised. Furthermore, it is able to learn translation-invariant novelty filters. The latter type of filter is able to separate familiar patterns from novel patterns. The abilities of the ASSOM makes it ideal for feature extraction and preprocessing in, for instance, an MPF.

When designing an MPF, it can prove vital to be able to limit the amount of data the MPF is subjected to. It is not desirable to simply apply raw data, from e.g. a video camera, directly to the MPF. Most often, it is most optimal to first preprocess the raw data and perform preliminary feature extraction, before processing the data in the MPF. As the MPF is the “bottleneck” the more we can limit the amount of data it has to process, the better (although there are limits).

Despite the powerfulness of the ASSOM, it also has problems. When applying the ASSOM, it is necessary to implement an ASSOM for each invariant feature the ASSOM has to learn. This means that one ASSOM cannot learn all of the different types of invariance at once. This can be problematic depending on the application. Therefore, it is likely that if ASSOMs should be used in conjunction with an MPF, their role will most likely be confined to preprocessing at the input level. Using ASSOMs at all the other layers of the MPF would likely cause a combinatorial explosion of the numbers of ASSOMs in the MPF. However, it appears that ASSOMs are very powerful unsupervised feature extraction filters if they are applied correctly.
Appendix E

Miscellaneous

E.1 Glossary

In the following two pages, a glossary of the most relevant words of this report is provided.

E.2 Version

This report is internal version A042.
<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Topology</td>
<td>Term coined by this author representing a model that is able to adapt a topology of nodes to input data</td>
</tr>
<tr>
<td>Bayesian network</td>
<td>Probabilistic graph model representing variables and their relationships</td>
</tr>
<tr>
<td>Bayesian Analysis</td>
<td>The application of Bayes’ theorem to perform inference and prediction</td>
</tr>
<tr>
<td>Belief propagation</td>
<td>Algorithm for inferring posteriors of nodes in graph and propagating beliefs</td>
</tr>
<tr>
<td>Chinese Restaurant Process</td>
<td>An example of a Dirichlet Process sampling scheme. Used to construct DP distributions based on Gibbs sampling</td>
</tr>
<tr>
<td>DHHMM</td>
<td>Dean’s version of the Hierarchical Hidden Markov Model (Dean 2006)</td>
</tr>
<tr>
<td>DHNBP</td>
<td>Dynamic HNBP. Dynamic Markov Network modeling sequences of invariant features based on NBP <em>(Chapter 7)</em></td>
</tr>
<tr>
<td>Dirichlet Process</td>
<td>A discrete stochastic probability distribution over distributions. The DP thus clusters continuous data</td>
</tr>
<tr>
<td>Explaining away</td>
<td>Confirming one variable reduces the likelihood of any other related variables (Pearl 1988)</td>
</tr>
<tr>
<td>Generative model</td>
<td>Data is modeled using a hidden variable. A Posterior dist. is computed from a prior and likelihood</td>
</tr>
<tr>
<td>Graph model</td>
<td>Representation of relationships between objects (vertices) by using links (edges or arcs).</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>In this report: A hierarchy is an ordered set of interconnected nodes or layers</td>
</tr>
<tr>
<td>Hierarchical scaling</td>
<td>The ordering of interconnected layers of different spatiotemporal scale</td>
</tr>
<tr>
<td>HNBP</td>
<td>Hierarchical NBP <em>(Chapter 6)</em>. A hierarchical expansion of the NBP framework.</td>
</tr>
<tr>
<td>HTM</td>
<td>Hierarchical Temporal Memory (George &amp; Hawkins 2005). An implementation of the MPF</td>
</tr>
<tr>
<td>Inference</td>
<td>In this report: The process determining the posterior distribution of an r.v.</td>
</tr>
<tr>
<td>Invariance</td>
<td>In this report: A representation of an object that does not change despite object is transformed in the input</td>
</tr>
<tr>
<td>Layers</td>
<td>In this report: A layer is a collection of nodes that are all on the same level of spatiotemporal scale</td>
</tr>
<tr>
<td>Likelihood</td>
<td>The probability of observing some data given the parameters of a model (used in generative models)</td>
</tr>
<tr>
<td>Markov chain</td>
<td>Stochastic process with the Markov property</td>
</tr>
</tbody>
</table>

Table E.1: Glossary Part 1
<table>
<thead>
<tr>
<th>Term</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Markov property</td>
<td>Markov chains where future states are independent of previous states given the current state</td>
</tr>
<tr>
<td>MCMC</td>
<td>Markov Chain Monte Carlo, a set of methods for approximating distributions</td>
</tr>
<tr>
<td>Mixture model</td>
<td>A convex combination of parametric parts. Thus, the resulting model is defined by mixture components</td>
</tr>
<tr>
<td>Monte Carlo integration</td>
<td>A stochastic approach to determining integrals by approximation</td>
</tr>
<tr>
<td>MPF</td>
<td>Memory-prediction framework, a model of the human brain (neocortex and others) by Hawkins et al. (2004)</td>
</tr>
<tr>
<td>NBP</td>
<td>Nonparametric Belief Propagation algorithm by Sudderth et al. (2003)</td>
</tr>
<tr>
<td>Nonparametrics</td>
<td>In nonparametrics there is no a priori assumption of what type of distribution the data originates</td>
</tr>
<tr>
<td>Node</td>
<td>In this report: Fundamental unit of a graph model, often used to represent a random variable</td>
</tr>
<tr>
<td>Overlapping Receptive Fields</td>
<td>Nodes higher in the hierarchy share nodes lower in the hierarchy</td>
</tr>
<tr>
<td>Particle Filtering</td>
<td>Stochastic approach to performing sequential Bayesian filtering. Distributions are comprised by particles</td>
</tr>
<tr>
<td>Posterior distribution</td>
<td>Conditional probability of a variable given some evidence</td>
</tr>
<tr>
<td>Potential function</td>
<td>In this report: The compatibility between two nodes</td>
</tr>
<tr>
<td>Prior distribution</td>
<td>Marginal probability of variable in absence of evidence</td>
</tr>
<tr>
<td>Sequential MCMC</td>
<td>Particle Filtering, MCMC sampling method based on sequences of data</td>
</tr>
<tr>
<td>Semi-Markov Process</td>
<td>Similar to a hidden Markov model only with the inclusion of a secondary dependency between states.</td>
</tr>
<tr>
<td>Slow Feature Analysis</td>
<td>Features relevant to learning invariant representations changes slowly compared to the sensory input</td>
</tr>
<tr>
<td>SOM</td>
<td>Self-Organizing Map</td>
</tr>
<tr>
<td>Spatiotemporal</td>
<td>In this report: Joint term for space and time. Used to denote a model that both accounts for space and time</td>
</tr>
<tr>
<td>Stochastic</td>
<td>Random or non-deterministic, in this report often used to indicate a non-deterministic part of a process</td>
</tr>
<tr>
<td>Topology</td>
<td>In this report: A specific arrangement of a set of nodes in a graph (not the same as mathematical topology)</td>
</tr>
</tbody>
</table>

Table E.2: Glossary Part 2
Bibliography


Dean, T. (2005), A Computational Model of the Cerebral Cortex, in ‘American Association for Artificial Intelligence’.


Murphy, K. P. (2001), An introduction to graphical models, Technical report, Department of Computer Science, University of British Columbia.

Murphy, K. P. (2002), Hidden semi-Markov models (HSMMs), Technical report, Department of Computer Science, University of British Columbia.


Numenta (2007), Problems that Fit HTM v. 1.0, Technical report, Numenta Inc.


Yin, H. & Allinson, N. M. (1998), Comparison of a Bayesian SOM with the EM Algorithm for Gaussian Mixtures, Technical report, UMIST, Dept. of Electrical Engineering and Electronics, Manchester, PO Box 88, M60 1QD, UK.