Helsinki University of Technology
Dissertations in Computer and Information Science
Espoo 2006 Report D18

Bayesian Inference in Nonlinear and Relational Latent Variable Models

Tapani Raiko

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering for public examination and debate in Auditorium T1 at Helsinki University of Technology (Espoo, Finland) on the 1st of December, 2006, at 12 o'clock noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory of Computer and Information Science

Distribution:
Helsinki University of Technology
Laboratory of Computer and Information Science
P.O. Box 5400
FI-02015 TKK
FINLAND
Tel. +358-9-451 3272
Fax +358-9-451 3277
http://www.cis.hut.fi

Available in PDF format at http://lib.tkk.fi/Diss/2006/isbn951228510X/

Printed version:
ISBN-13 978-951-22-8509-9
ISBN-10 951-22-8509-6

Electronic version:
ISBN-13 978-951-22-8510-5
ISBN-10 951-22-8510-X

ISSN 1459-7020

Otamedia Oy
Espoo 2006

Raiko, T. (2006): **Bayesian Inference in Nonlinear and Relational Latent Variable Models**. Doctoral thesis, Helsinki University of Technology, Dissertations in Computer and Information Science, Report D18, Espoo, Finland.

Keywords: machine learning, graphical models, probabilistic reasoning, nonlinear models, variational methods, state-space models, hidden Markov models, inductive logic programming, first-order logic

ABSTRACT

Statistical data analysis is becoming more and more important when growing amounts of data are collected in various fields of life. Automated learning algorithms provide a way to discover relevant concepts and representations that can be further used in analysis and decision making.

Graphical models are an important subclass of statistical machine learning that have clear semantics and a sound theoretical foundation. A graphical model is a graph whose nodes represent random variables and edges define the dependency structure between them. Bayesian inference solves the probability distribution over unknown variables given the data. Graphical models are modular, that is, complex systems can be built by combining simple parts. Applying graphical models within the limits used in the 1980s is straightforward, but relaxing the strict assumptions is a challenging and an active field of research.

This thesis introduces, studies, and improves extensions of graphical models that can be roughly divided into two categories. The first category involves nonlinear models inspired by neural networks. Variational Bayesian learning is used to counter overfitting and computational complexity. A framework where efficient update rules are derived automatically for a model structure given by the user, is introduced. Compared to similar existing systems, it provides new functionality such as nonlinearities and variance modelling. Variational Bayesian methods are applied to reconstructing corrupted data and to controlling a dynamic system. A new algorithm is developed for efficient and reliable inference in nonlinear state-space models.

The second category involves relational models. This means that observations may have distinctive internal structure and they may be linked to each other. A novel method called logical hidden Markov model is introduced for analysing sequences of logical atoms, and applied to classifying protein secondary structures. Algorithms for inference, parameter estimation, and structural learning are given. Also, the first graphical model for analysing nonlinear dependencies in relational data, is introduced in the thesis.

Raiko, T. (2006): **Bayesiläinen päättely epälineaarisissa ja rakenteisissa piilomuuttujamalleissa**. Tohtorin väitöskirja, Teknillinen korkeakoulu, Dissertations in Computer and Information Science, raportti D18, Espoo, Suomi.

Avainsanat: koneoppiminen, graafiset mallit, todennäköisyyslaskentaan perustuva päättely, epälineaariset mallit, variaatiomenetelmät, tila-avaruusmallit, piilo-Markov -malli, induktiivinen logiikkaohjelmointi, ensimmäisen kertaluvun logiikka

TIIVISTELMÄ

Tilastollisen tietojenkäsittelyn merkitys on vahvassa kasvussa, sillä tietoaineistoa kerätään yhä enemmän lukuisilla eri aloilla. Automaattisilla oppivilla menetelmillä voidaan löytää merkityksellisiä käsitteitä ja esitysmuotoja, joita voidaan edelleen käyttää analysoinnissa ja päätöksenteossa.

Tärkeä tilastollisen koneoppimisen menetelmäperhe, graafiset mallit, on selkeästi tulkittavissa ja sillä on hyvä teoreettinen perusta. Graafinen malli koostuu verkosta, jonka solmut kuvaavat satunnaismuuttujia ja linkit määrittelevät niiden väliset riippuvuussuhteet. Bayesiläinen päättely ratkaisee tuntemattomien muuttujien jakauman aineiston ehdolla. Graafiset mallit ovat modulaarisia, eli monimutkaisia järjestelmiä voidaan rakentaa yhdistelemällä yksinkertaisia osia. 1980-luvun tiukkojen oletusten puitteissa graafisten mallien soveltaminen on suoraviivaista, mutta näiden oletusten väljentäminen on haastava ja aktiivinen tutkimuskohde.

Tässä väitöstyössä esitellään, tutkitaan ja parannellaan uusia graafisten mallien laajennuksia, jotka voidaan karkeasti jakaa kahteen luokkaan. Ensimmäiseen luokkaan kuuluvat neuroverkkojen inspiroimat epälineaariset mallit, joissa sovelletaan bayesiläistä variaatio-oppimista ylioppimisen ja laskennallisen vaativuuden välttämiseen. Työ esittelee kehyksen, jossa käyttäjän antaman mallin tehokkaat päivityssäännöt ratkaistaan automaattisesti. Vastaaviin järjestelmiin verrattuna se tarjoaa uusia toimintoja, kuten epälineaarisuuksia ja hajonnan mallinnusta. Bayesiläisiä variaatiomenetelmiä käytetään viallisen tietoaineiston rekonstruointiin ja dynaamisen systeemin säätöön. Uusi algoritmi hoitaa epälineaaristen tila-avaruusmallien päättelyn tehokkaasti ja luotettavasti.

Toinen laajennusten luokka käsittelee relaatiomalleja, joissa havainnoilla voi olla vaihteleva sisäinen rakenne ja viittauksia toisiinsa. Uusi menetelmä, looginen piilo-Markov -malli, esitellään loogisten atomien sarjojen analysointiin ja sitä sovelletaan proteiinien sekundäärirakenteen luokitteluun. Menetelmälle esitetään algoritmit päättelyyn, parametrien määritykseen ja rakenteen oppimiseen. Työssä esitellään myös ensimmäinen graafinen malli relaatioaineistojen epälineaaristen riippuvuussuhteiden analysointiin.

Preface

This work has been carried out at the Laboratory of Computer and Information Science in Helsinki University of Technology and at the Laboratory of Machine Learning and Natural Language Processing in University of Freiburg. Other sources of funding were the Graduate School in Computational Methods of Information Technology (ComMIT), the Finnish Centre of Excellence Programme (2000-2005) under the project New Information Processing Principles, the European Commission's IST-funded projects BLISS (IST-1999-14190), the European Commission's IST-funded Network of Excellence for Multimodal Interfaces PASCAL (IST-2002-506778), the European Commission's IST-funded evaluation project APRIL (IST-2001-33053), the Finnish Cultural foundation, and the Nokia Foundation.

I wish to thank my instructor Dr. Harri Valpola for inspiration and guidance, especially in encouraging me to reach high. I also wish to thank my supervisor Prof. Juha Karhunen for his dedication and support, especially for keeping my feet on the ground. This experience has allowed me grow as a person.

I wish to express my gratitude to the co-authors of the publications of the thesis, Dr. Kristian Kersting, Prof. Dr. Luc De Raedt, Matti Tornio, Dr. Antti Honkela, Markus Harva, Tomas Östman, and Prof. Dr. Stefan Kramer. I also wish to thank my other coworkers in the laboratories, including Prof. Erkki Oja, Dr. Jaakko Peltonen, Dr. Alexander Ilin, and Dr. Sampsa Laine for help and interesting discussions as well as my pre-examiners Prof. Jouko Lampinen and Prof. Petri Myllymäki for useful comments.

Last but not least, I thank Anna Hiironen for her support and help, as well as for encouraging me to work abroad.

Espoo, November 2006

Tapani Raiko

Contents

Publications of the thesis

I T. Raiko, H. Valpola, M. Harva, and J. Karhunen. Building Blocks for Variational Bayesian Learning of Latent Variable Models. Report E4 in the Electronic report series of CIS, April, 2006, accepted for publication conditioned on minor revisions to the Journal of Machine Learning Research.

II T. Raiko, H. Valpola, T. Östman, and J. Karhunen. Missing Values in Hierarchical Nonlinear Factor Analysis. In the Proceedings of the International Conference on Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP 2003), pp. 185–189, Istanbul, Turkey, June 26–29, 2003.

III T. Raiko. Partially Observed Values. In the Proceedings of the International Joint Conference on Neural Networks (IJCNN 2004), pp. 2825–2830, Budapest, Hungary, July 25–29, 2004.

IV T. Raiko and M. Tornio. Learning Nonlinear State-Space Models for Control. In the Proceedings of the International Joint Conference on Neural Networks (IJCNN 2005), pp. 815–820, Montreal, Canada, July 31–August 4, 2005.

V T. Raiko, M. Tornio, A. Honkela, and J. Karhunen. State Inference in Variational Bayesian Nonlinear State-Space Models. In the Proceedings of the 6th International Conference on Independent Component Analysis and Blind Source Separation (ICA 2006), pp. 222–229, Charleston, South Carolina, USA, March 5–8, 2006.

VI T. Raiko. Nonlinear Relational Markov Networks with an Application to the Game of Go. In the Proceedings of the International Conference on Artificial Neural Networks (ICANN 2005), pp. 989–996, Warsaw, Poland, September 11–15, 2005.

VII K. Kersting, L. De Raedt, and T. Raiko. Logical Hidden Markov Models. In the Journal of Artificial Intelligence Research, Volume 25, pp. 425–456, April, 2006.

VIII K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards Discovering Structural Signatures of Protein Folds based on Logical Hidden Markov Models. In the Proceedings of the Pacific Symposium on Biocomputing (PSB-2003), pp. 192–203, Kauai, Hawaii, January 3–7, 2003.

IX K. Kersting and T. Raiko. 'Say EM' for Selecting Probabilistic Models for Logical Sequences. In the Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005), pp. 300–307, Edinburgh, Scotland, July 26–29, 2005.

List of abbreviations

AI	Artificial intelligence
BIC	Bayesian information criterion
BLP	Bayesian logic program
BP	Belief propagation (algorithm)
EM	Expectation maximisation
FA	Factor analysis
HNFA	Hierarchical nonlinear factor analysis
HMM	Hidden Markov model
ICA	Independent component analysis
ILP	Inductive logic programming
KL	Kullback–Leibler (divergence)
LOHMM	Logical Hidden Markov model
MAP	Maximum a posteriori (estimate)
ML	Maximum likelihood (estimate)
MCMC	Markov chain Monte Carlo
MLP	Multilayer perceptron (network)
NDFA	Nonlinear dynamic factor analysis
NMN	Nonlinear Markov network
NRMN	Nonlinear relational Markov network
NSSM	Nonlinear state-space model
pdf	Probability density function
PoE	Product of experts
PCA	Principal component analysis
PRM	Probabilistic relational model
RMN	Relational Markov network
SRL	Statistical relational learning
VB	Variational Bayesian

List of symbols

$\wedge$	And
$\neg$	Negation
A, B, C	Variables, events, or actions
x, y, z	Scalar variables
$P(A \mid B)$	Probability of A given B
$p(A \mid B)$	Probability density of A given B
$\boldsymbol{X}$	Observations (or data)
$\mathbf{\Theta}$	Unknown variables $\mathbf{\Theta} = (\boldsymbol{\theta}, \boldsymbol{S})$
$\boldsymbol{\theta}$	Model parameters $\boldsymbol{\theta}$
$\boldsymbol{S}$	Latent variables
$U(A)$	Utility of A
$\mathcal{H}$	Model structure and prior belief
$\mathcal{N}(x; y, z)$	Gaussian distribution of x with a mean y and a variance z
$\propto$	Proportional to (or equals after normalisation)
π	Message sent away from root (belief propagation algorithm)
λ	Message sent towards the root (belief propagation algorithm)
ψ	Potential in a Markov network
$q(\mathbf{\Theta})$	Approximation of the posterior distribution $p(\mathbf{\Theta} \mid \boldsymbol{X})$
$D(q \parallel p)$	Kullback-Leibler divergence between q and p
$\mathbf{x}(t)$	Observation (or data) vector for (time) index t
$\mathbf{s}(t)$	Source (or factor) vector for (time) index t
$\mathbf{u}(t)$	Auxiliary vector (either for control or variance modelling)
$\mathbf{f}$	Mapping from the source space to the observation space
$\mathbf{g}$	Mapping for modelling dynamics in the source space
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$	Matrices belonging to parameters $\boldsymbol{\theta}$
$\overline{\theta}$	Mean of the parameter θ in the approximating posterior distribution q
$\widetilde{\theta}$	Variance of the parameter θ in the approximating posterior distribution q
$\langle \cdot \rangle$	Expectation over the distribution q
X, Y, Z	Logical variables
$\leftarrow$	Follows from (in logic programming)
X	Observed sequence of logical atoms

Chapter 1

Introduction

Statistical machine learning aims at discovering relevant concepts and representation of data collected in various fields of life. Learned models can be used to analyse and summarise the data, to reconstruct missing information and predict future data, to make decisions, plan and control. There has been huge research activity covering various tasks in various applications leading to a diverse collection of methods.

Let us consider an example of intensive care unit which is a hospital bed equipped for medical care and observation to people in a critical or unstable condition. Hanson and Marshall (2001) note that the intensive care environment is particularly suited to the implementation of artificial intelligence tools because of the wealth of available data and the inherent opportunities for increased efficiency in inpatient care. There are about 250 variables online, daily laboratory data, and relational background data including care history, nutrition, infections, relatives, and demographic data. In principle, there are machine learning methods that could be used to learn from previous patients and applied to new patients. In practice, it is very difficult to take the wealth of information into account because of the diversity of data and applicable methods. Similar situation applies in other application areas from robotics to economical modelling.

Graphical models (Pearl, 1988; Jensen et al., 1990; Cowell et al., 1999; Neapolitan, 2004; Bishop, 2006) are an important subclass of statistical machine learning methods that have clear semantics and a sound theoretical foundation. A graphical model is a graph whose nodes represent random variables and edges define the dependency structure between them. Bayesian inference solves the probability distribution over unknown variables given the data. Many methods in machine learning that are not originally graphical models, can be reinterpreted or transformed into the framework. This allows one to combine different methods in a principled manner, as well as to reuse ideas and software between sometimes surprisingly different applications.

Latent variable models aim at explaining the observed data by supplementing it with unknown factors or a hidden state. The idea is that even if the regularities in the data itself are difficult to find, the dependencies between latent variables and observations are simpler, given that a proper representation is found. Model parameters and latent variables can be solved at the same time in the framework of graphical models.

Basic tasks in graphical models, such as inference and learning, have been solved for decades, but relaxing the strict assumptions such as linearity of the dependencies or that the data comes in uniform samples, is a challenging and an active field of research. This thesis studies and introduces several extensions to the well-known existing graphical models.

This thesis consists of an introductory part, whose structure is shown in Figure 1.1, and nine publications described in Section 1.3.

1.1 Background

Expert systems (see for example the book by Giarratano and Riley, 1994) were popular in the artificial intelligence (AI) community in the 70s. They consist of simple rules of thumb in the form of *if... then...*, such as *if burglar or earthquake then alarm*. A specialist in the field constructs a number of rules for a narrow problem domain, and an inference engine could apply the rules to an initial set of facts to obtain answers. The rules form a network where the output of a rule can be used as an input for another rule. In the simplest case, the rules are restricted to propositional calculus and truth values are binary, that is, simple statements are either true or false. Many methods in the field of AI and machine learning can be seen as extensions of expert systems in different directions.

In some cases the chain of reasoning from the initial facts to answers is very long. If there is a constant number of options at each step, the size of the solution space grows exponentially and the problem becomes intractable. Chess is a typical example of such a problem. Searching (and planning) (see book by Russell and Norvig, 1995) aims at finding the optimal solution as fast as possible, and finding

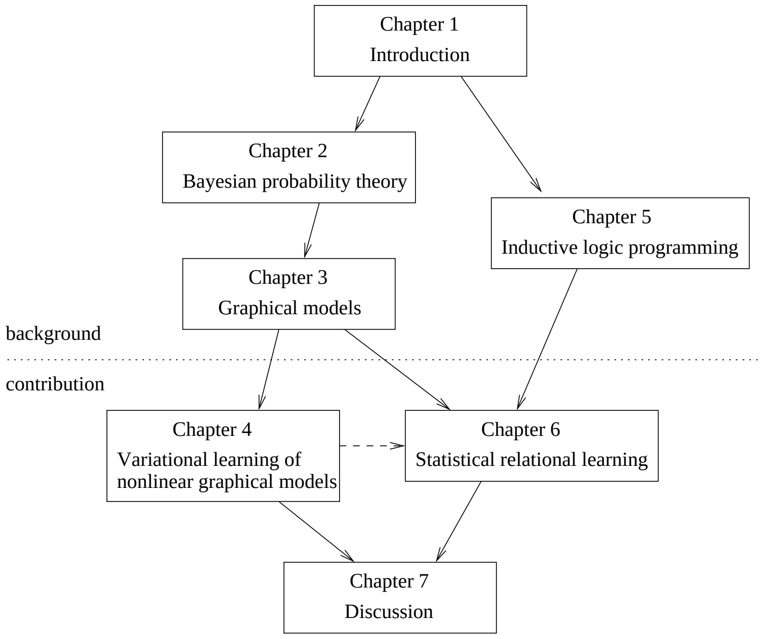


Figure 1.1: Dependency structure of the chapters of the thesis.

a reasonable solution in case the problem is simply too large. In this thesis, the reasoning chains are small enough so that the full structure can be always explored, but search appears in the space of solution structures, that is, on a different level of abstraction.

Fuzzy logic (see book by Klir and Yuan, 1995) extends the truth values of expert systems into a continuum between 0 and 1, for instance an apple might be *somewhat* red or a person might be *somewhat* asleep. Fuzzy logic is an internally consistent body of mathematical tools but fuzzy truth values should not be interpreted as measures of uncertainty (Dubois and Prade, 1993). For instance, assume that the truth value of the event A, catching the bus, is 0.5, then the truth values of $A \wedge A$ and $A \wedge \neg A$ are the same in fuzzy logic. One can imagine a person being both somewhat asleep and somewhat awake at the same time, but somewhat catching a bus does not make sense. This simple example shows that fuzzy logic alone is not enough in an uncertain world. Section 4.1.5 discusses how fuzzy concepts can, in some sense, be brought into the probabilistic framework instead.

Replacing truth values of expert systems with probabilistic variables leads to graphical models (Pearl, 1988; Neapolitan, 2004; Bishop, 2006). Graphical models can perform inferences such as "even though I heard an alarm, the probability of a burglar entering the house is fairly small because I noticed an earthquake that can also trigger the alarm." Most expert systems would have problems using rules in both directions like in this case. Graphical models combine graph theory with probability theory. Both the structure of the graph and the parameters determining the probabilities can be learned from data. Graphical models form the basis for this work so they will be described in Chapter 3.

Sometimes facts are viewed as states and rules are viewed as actions. In some cases things do not change much, for example searching becomes planning with exactly the same algorithms. Graphical models generalise to influence diagrams (Pearl, 1988). The most important concern is to keep track of what information is available to the decision maker at the time of decision. The goal of the decision maker is to maximise *utility* that it receives after the decisions. Decision theory is reviewed in Section 2.4.

Perhaps the best known artificial neural network, the multi-layer perceptron (MLP) network (Haykin, 1999), can be related to expert systems, too. MLP network concentrates on a single *if* $\mathbf{x}$ *then* $\mathbf{y}$ rule where $\mathbf{x}$ and $\mathbf{y}$ are vectors of real values. The task is to learn a nonlinear dependency based on data. An MLP network can be constructed as a graphical model with nonlinear dependencies, allowing for new functionality, as described in Chapter 4.

Latent variable models assume unknown source signals (also called factors, latent or hidden variables, or hidden causes) to have generated the observed data through an unknown mapping or process. The goal of learning is to identify both the source signals and the unknown generative mapping (or process). The success of a specific model depends on how well it captures the structure of the phenomena underlying the observations. Various linear models have been popular (see Hyvärinen et al., 2001), because their mathematical treatment is fairly easy. However, in many realistic cases the observations have been generated by a nonlinear process. Learning of a nonlinear latent variable model is a challenging task, because it is computationally much more demanding and flexible models require also strong regularisation. Variational Bayesian methods, described in Section 4.1, qualify for both computational efficiency and regularisation.

Yet another direction to extend basic expert systems is to replace propositional calculus by first-order logic. The results are still called expert systems or logic programmes. One of the inference engines for first-order logic is Prolog (Sterling and Shapiro, 1994). It is also possible to learn logic programmes from relational data. This is called inductive logic programming (Lavrac and Dzeroski, 1994; De Raedt, 2005). First-order logic allows for handling rich internal structure such as samples with a varying internal structure or links between samples. Logic programming and inductive logic programming are described in Chapter 5 and further combined with graphical models in Chapter 6.

1.2 Contributions of the thesis

Graphical models provide a good framework for machine learning and artificial intelligence. Graphical models are going to be extended, based on approximate Bayesian inference, into a system that can plan, infer, and interact with its environment using both discrete and continuous variables as well as structured representations. The concrete steps that have been taken in this work can be summarised as follows:

- A novel framework where Bayesian networks may include nonlinear dependencies and algorithms for variational Bayesian learning are automatically derived.
- An extension of hidden Markov models to deal with sequences of structured symbols rather than characters, with four relevant algorithms and an applications in the domain of bioinformatics.
- The first graphical model that can handle both nonlinear dependencies and relations.
- An extension of a method for learning nonlinear state-space models to control.
- A novel algorithm for inference in nonlinear state-space models that is both efficient and reliable.
- A study of methods for handling corrupted or inaccurate values in data.
- A study of some latent variable models based on their capability of reconstructing missing values in data.

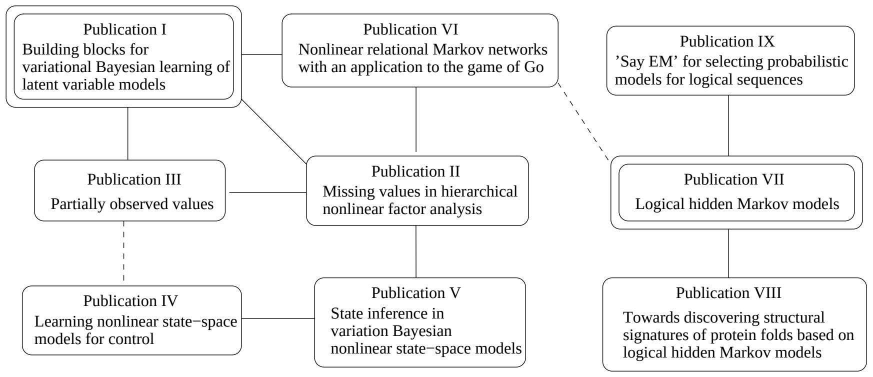

Publication I
Building blocks for variational Bayesian learning of latent variable models
Publication VI
Nonlinear relational Markov networks with an application to the game of Go
Publication IX
'Say EM' for selecting probabilistic models for logical sequences
Publication III
Partially observed values
Publication II
Missing values in hierarchical nonlinear factor analysis
Publication VII
Logical hidden Markov models
Publication IV
Learning nonlinear state−space models for control
Publication V
State inference in variation Bayesian nonlinear state−space models
Publication VIII
Towards discovering structural signatures of protein folds based on logical hidden Markov models

Figure 1.2: Publications of the thesis. Journal articles have two frames and conference papers just one. Relationships are shown as edges that are undirected since there is no clear causality.

1.3 Contents of the publications and author's contributions

The titles of the nine publications of this thesis and their relationships are shown in Figure 1.2.

Publication I introduces a framework for creating graphical models from simple building blocks. It is based on variational Bayesian learning, and unlike other such frameworks, it can model nonlinearities and nonstationary variance. Once the user defines the model structure, the algorithms for learning and inference are automatically derived. The present author developed a part of the framework, carried out a small part in the implementation, made two of the three experiments and wrote a large part of the paper.

Well-founded handling of missing values is one of the advantages of Bayesian modelling. Publication II studies the reconstruction of missing values in nonlinear factor analysis. The present author made the implementation, ran the experiments, and wrote most of the paper under the guidance of Dr. Harri Valpola.

Values in the data are often either observed, or missing, but some cases fall in between: Sometimes it is known that a measurement is inaccurate, or perhaps there is only a lower bound. Publication III studies handling and reconstruction of such partially observed values in the variational Bayesian framework. It also brings up a situation where the cost function of the variational Bayesian learning can diverge to negative infinity. It can be solved using partially observed values or by adding virtual noise in the data.

Publication IV applies a state-of-the-art method from machine learning to the problem of nonlinear model-predictive control. Three different control schemes are studied, one is based directly on the learned neural network, the second one is the traditional nonlinear model-predictive control, and the third one is based on Bayesian inference. The present author designed the novel control scheme and wrote a large part of the paper.

The control application brought up a setting to which none of the tested inference algorithms for nonlinear state-space models suited well. Publication V introduces a novel algorithm that both converges reliably and is still fast. The present author designed the algorithm and wrote a large part of the paper.

The last four publications involve relations. Publication VI gives the first extension of graphical models to both nonlinear and relational direction at the same time. The relations in the data define a structure for a graphical model where unknown variables can then be inferred using variational Bayesian methods. The novel method is applied to the analysis of the board game Go.

Hidden Markov models (HMMs) are very popular for analysing sequential data. Logical hidden Markov models (LOHMMs) extend traditional hidden Markov models to deal with sequences of structured symbols in the form of logical atoms, rather than characters. Publication VII formally introduces LOHMMs and presents efficient solutions to the three central inference problems for LOHMMs: evaluation, most likely hidden state sequence and parameter estimation. The idea came from Prof. Luc De Raedt whereas Dr. Kristian Kersting and the present author jointly formalised and implemented the LOHMMs. The present author's contribution in experimentation and writing were minor.

In Publication VIII, LOHMMs are applied to the domain of bioinformatics. The task was to extract structural signatures of folds for classes of proteins according to the classification scheme SCOP. The results indicate that LOHMMs possess several advantages over other methods. The present author took part in the design, implementation, experimentation, and writing.

The increase of descriptive power of LOHMMs over HMMs comes at the expense of a more complex model selection problem, since different abstraction levels need to be explored. Publication IX presents a novel algorithm for choosing the model structure. The effectiveness of the algorithm is confirmed both theoretically and by experimentation with real-world unix command sequence data. The work was done jointly by Dr. K. Kersting and the present author.

Chapter 2

Bayesian probability theory

Bayesian probability theory (Jaynes, 2003; MacKay, 2003; Pearl, 1988) defines probabilities to be subjective. Probabilities measure the credibility of an event so they can depend on the subjects prior knowledge, and they are updated based on observations. Say, you toss a coin and cover it with your hand without looking. The probabilities of heads or tails are even. When you peek under your hand, only the information available for you changes. For other people, the chances are still even.

Bayesian probability theory gives a well-founded methodology for handling uncertainty. Given a model that describes the mutual dependencies of random variables, Bayesian probability theory can then be used to infer all the unknown variables. This chapter gives an introduction to the theory and to practicalities of Bayesian treatment of uncertainty from the machine-learning point of view.

2.1 Representations of data and belief

This thesis deals with three types of representational elements: discrete values, continuous values, and relations. This applies to both internal beliefs of the system and observations (or data). Other types of data should be possible to convert to them in a more or less sensible manner. Anderberg (1973) discusses the representations in detail, excluding relations.

For categorical variables, only a finite number of values is possible. For instance, the blood type of a person is one of four possibilities. A coin toss has two possibilities. The alphabet has 26 letters. Text is often processed by giving a discrete label to each known word.

The measured sound pressure in a room is an example of a continuous value or a real number. Most physical measurements come as continuous values, such as measuring the time, weight, length, or temperature. Also the sensory systems in living organisms and robots produce continuous values. Digital cameras and scanners convert images into data where the image is divided into small square pixels that have a constant colour. The colours are described by three numbers, the red, green, and blue intensities. Sound waves can be represented as a sequence of air pressure values, like in CDs. Note that even though values are always represented with limited accuracy in computers, in theory they are handled as real numbers.

Discrete numerical variables, such as the number of children, can be processed as categorical data by making a finite number of categories such as 0,1,2,3,4,5+. The other option is to reinterpret the ordinal value as a continuous value. This is often done by people, too. We have no difficulties in understanding a statement such as "Finnish women give birth to 1.7 children on average". Section IV B of Publication III studies and solves a problem originating from a conversion of discrete to continuous values.

The third type of representation, relations, is rather different from the other two. Relations are used to relate objects to one another. Codd (1970) wrote a significant paper about general relational databases. The basic idea is that access to the data is unaffected by the internal representation. This becomes important when more and more different types of data are integrated together into a common databank. Relational databases have become a standard. The universal model for data is basically a set of tables where different columns are different attributes that can be of varying type, and rows are objects. Values in the table may include identifiers that point to other rows of the same or another table. For example, a molecule can be represented as two tables, where the first one lists all atoms with their identifiers and attributes, and the second table lists all bonds, with identifiers of the involved atoms and the attributes of the bond. Similar representation applies to web pages, where instead of bonds, the second table lists links.

As the biological senses never produce identifiers directly, they have to be created by the mind. For example when a predator tries to catch its pray by wearing it down, it is important for the chaser to stick to the same target even if it cannot be recognised from the herd. Also, to know the structure of a molecule, one has to know which atoms are connected with bonds even if the atoms as such are indistinguishable. Both cases can be solved by giving an implicit or explicit identifier for the prey or atom. Pointers are the identifiers used in computer code to refer to different parts of the memory, and thus to different objects.

In Bayesian analysis, the belief or uncertainty about variables is represented with probabilities. The probability of an event A given prior knowledge B is written as $P(A \mid B)$. Similar notation can be used when A is a discrete variable: $P(A \mid B)$ denotes the probability distribution of A given B. Continuous probability distribution can be represented with a probability density function (pdf) $p(\cdot)$. The actual probability is an integral over the pdf. It is also called probability mass, using an analogy from physics. For example the probability of an event $A < 0$ given B can be computed as $P(A < 0 \mid B) = \int_{-\infty}^{0} p(A|B)dA$.

The rest of the chapter is written for continuous values, but rewriting the integrals as sums produces the corresponding formulas for discrete values. The treatment of relations is left to Chapters 5 and 6.

2.2 The Bayes rule and the marginalisation principle

The Bayes rule was formulated by reverend Thomas Bayes in the 18th century (Bayes, 1958). It can be derived from very basic axioms (Cox, 1946). The Bayes rule tells how to update ones beliefs when receiving new information. In the following, $\mathcal{H}$ stands for the assumed model, $\boldsymbol{X}$ stands for observation (or data), and $\boldsymbol{\Theta}$ stands for unknown variables. $p(\boldsymbol{\Theta} \mid \mathcal{H})$ is the prior distribution, or the distribution of the unknown variables before making the observation. The posterior distribution is

$$p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H}) = \frac{p(\boldsymbol{X} \mid \mathcal{H}, \boldsymbol{\Theta}) p(\boldsymbol{\Theta} \mid \mathcal{H})}{p(\boldsymbol{X} \mid \mathcal{H})}. \tag{2.1}$$

The term $p(\boldsymbol{X} \mid \mathcal{H}, \boldsymbol{\Theta})$ is called the likelihood of the unknown variables given the data and the term $p(\boldsymbol{X} \mid \mathcal{H})$ is called the evidence (or marginal likelihood) of the model.

The marginalisation principle specifies how a learning system can predict or generalise. The probability of observing A with prior knowledge of $\boldsymbol{X}, \mathcal{H}$ is

$$p(A \mid \boldsymbol{X}, \mathcal{H}) = \int p(A \mid \boldsymbol{\Theta}, \boldsymbol{X}, \mathcal{H}) p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H}) d\boldsymbol{\Theta}. \tag{2.2}$$

It means that the probability of observing A can be acquired by summing or integrating over all different explanations $\boldsymbol{\Theta}$. The term $p(A \mid \boldsymbol{\Theta}, \boldsymbol{X}, \mathcal{H})$ is the probability of A given a particular explanation $\boldsymbol{\Theta}$ and it is weighted with the probability of the explanation $p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H})$.

Using the marginalisation principle, the evidence term can be written as

$$p(\boldsymbol{X} \mid \mathcal{H}) = \int p(\boldsymbol{X} \mid \boldsymbol{\Theta}, \mathcal{H}) p(\boldsymbol{\Theta} \mid \mathcal{H}) d\boldsymbol{\Theta}. \tag{2.3}$$

This emphasises the role of the evidence term as a normalisation coefficient. It is an integral over the numerator of the Bayes rule (2.1). Sometimes it is impossible to compute the integral exactly, but fortunately it is not always necessary. For example, when comparing posterior probabilities of different instantiations of hidden variables, the evidence cancels out.

2.3 Structure among unknown variables

For getting a model that is useful in new situations, i.e. having generalisation ability, some structure among the unknown variables $\boldsymbol{\Theta}$ needs to be assumed. A typical structure in machine learning is a division of unknown variables $\boldsymbol{\Theta}$ into parameters $\boldsymbol{\theta}$ and latent variables $\boldsymbol{S}$, $\boldsymbol{\Theta} = (\boldsymbol{\theta}, \boldsymbol{S})$. The distinction is that parameters $\boldsymbol{\theta}$ are shared among data samples, but there are separate latent variables for each data sample. Thus, the number of latent variables grows linearly with data size while the number of parameters stays the same. The latent variables can be thought of as the internal state of a system. Sometimes computing the posterior distribution over the parameters $\boldsymbol{\theta}$ is called Bayesian learning, leaving the term Bayesian inference to only refer to computing the posterior distribution of latent variables $\boldsymbol{S}$.

Graphical models, described in Chapter 3, provide a formalism for defining the exact structure of dependencies. The fundamental idea is that a complex system can be built by combining simpler parts. A graphical model is a graph whose nodes represent random variables and edges represent direct dependencies.

2.4 Decision theory

Decision theory (see book by Dean and Wellman, 1991) was first formulated by Blaise Pascal in the 17th century. When faced with a number of actions A, each with possibly more than one possible outcome B with different probabilities $P(B \mid A)$, the rational choice is the action that gives the highest expected value $U(A)$:

$$U(A) = \int_B U(A, B)P(B \mid A), \tag{2.4}$$

where $U(A, B)$ is the value of action A producing the outcome B. Daniel Bernoulli refined the idea in the 18th century. In his solution, he defines a utility function and computes expected utility rather than expected financial value. For example, people tend to insure their property even though the expected financial value of the decision is negative (after all, insurance business must be profitable for the insurance companies). This is explained by the fact that in case of losing one's whole property, every euro is more important (has more utility).

Bayesian decision theory (see book by Bernardo and Smith, 2000) works in situations where the outcomes of actions are unknown but one still has to make decisions. One simply chooses the action with highest expected utility over the predictive distribution of outcomes. Bayesian decision theory does not just give a way to make actions but it is actually the only coherent way of acting. This can be shown using a so-called Dutch book argument (Resnik, 1987). A Dutch book is a set of odds and bets which guarantees a profit, no matter what the outcome of the gamble. A Dutch book can never be made against a Bayesian decision maker, and if decisions are such that a Dutch book cannot be made against the decision maker, the decisions can always be interpreted to be made by a Bayesian decision maker.

Decision theory has a clear connection to control theory. (Dean and Wellman, 1991) The control that minimises a certain cost functional is called the optimal control (see book by Kirk, 2004). When the control cost is used as the negative utility $-U(\cdot)$, decision theory provides optimal control, even under uncertainty (stochastic control). Section 4.3.2 and Publication IV apply Bayesian decision theory for control in nonlinear state-space models.

2.5 Approximations

The Bayesian probability theory and decision theory sound too good to be true: They solve learning, inference, and decision making optimally. Unfortunately, the posterior probability distribution cannot be handled analytically except in the simplest examples. To solve the integrals in Equations (2.2), (2.3), and (2.4), one must resort to some kind of approximations. There are three common classes of approximations: point estimates, sampling, and variational approximations.

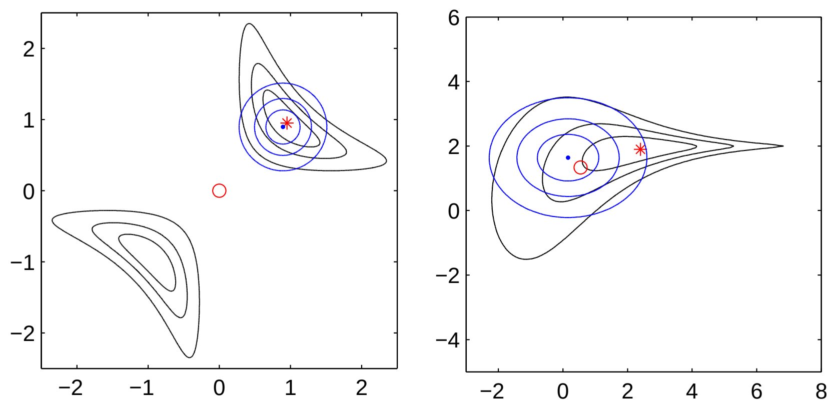


Figure 2.1: Posterior distributions of x and y are shown in black contours. Maximum a posterior estimate is plotted as a red star, Bayes estimate (or the expectation over the posterior) is plotted as a red circle. Variational Bayesian solution with a Gaussian posterior approximation with diagonal covariance is shown in blue as a dot surrounded by ellipses. Left: model $p(z) = \mathcal{N}(z; xy, 0.02)$, observation $z = 1$, priors $p(x) = \mathcal{N}(x; 0, 1)$, $p(y) = \mathcal{N}(y; 0, 1)$. Right: model $p(z) = \mathcal{N}(z; y, \exp(-x))$, observation $z = 2$, priors $p(x) = \mathcal{N}(x; -1, 5)$, $p(y) = \mathcal{N}(y; 0, 5)$.

Figure 2.1 shows two posterior distributions. The models are not particularly meaningful (having just two unknown variables), but they are chosen to highlight differences in various posterior approximations, which are described in the following.

2.5.1 Point estimates

Point estimates use a single representative value to summarise the whole posterior distribution. The maximum likelihood (ML) estimate (or solution) for the unknown variables $\boldsymbol{\Theta}$ is the point in which the likelihood $p(\boldsymbol{X} \mid \boldsymbol{\Theta}, \mathcal{H})$ is highest. The maximum a posteriori (MAP) estimate is the one with highest posterior probability density $p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H})$. Note that a common and even simpler criterion, the mean square error, is equivalent to the ML estimate assuming Gaussian noise with constant variance (Bishop, 1995).

An iterative learning algorithm is said to *overlearn* the training data set, when its

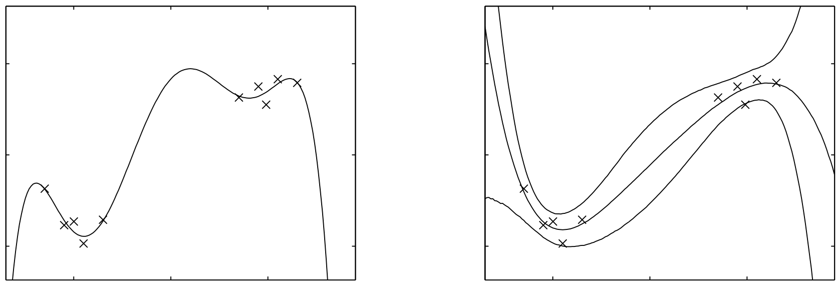


Figure 2.2: A sixth order polynomial is fitted to 10 data points. *Left:* Maximum likelihood solution. *Right:* Bayesian solution. The three curves present 5%, 50% and 95% fractiles.

performance with unseen test data starts to get worse during the learning with training data (Haykin, 1999; Bishop, 1995; Chen, 1990). The system starts to lose its ability to generalise. The same can happen when increasing the complexity of the model. The model is said to *overfit* to the data. In this case the model becomes too complicated and concentrates on random fluctuations in the data. The left subfigure of Figure 2.2 shows an example of overfitting.

When the model is too simple or the learning is stopped too early, the problem is called *underfitting* or *underlearning* respectively. Balancing between over- and underfitting has perhaps been the main difficulty in model building. There are several ways to fight overfitting and overlearning (Haykin, 1999; Bishop, 1995; Chen, 1990). A popular method is to select the best time to stop learning or the best complexity of a model by cross-validation (Stone, 1974; Haykin, 1999). Part of the training data is left for validation and the models are compared based on their performance with the validation set.

The problems of overlearning and overfitting are mostly related to point estimates. The example in Figure 2.2 is solved by using the whole posterior distribution instead of a single solution. The use of a point estimate is to approximate integrals so it should be sensitive to the probability mass rather than to the probability density. Unfortunately, ML and MAP estimates are attracted to high but sometimes narrow peaks. Figure 2.3 shows a situation, where search for the MAP solution first finds a good representative of the probability mass, but then moves to the highest peak which is on the border. This type of situation seems to be very common and the effect becomes stronger, when the dimensionality increases. Appendix D of

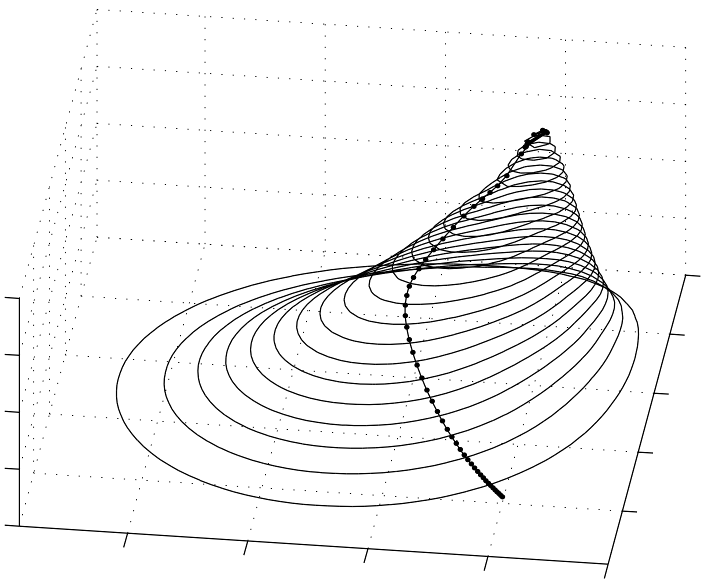


Figure 2.3: A hypothetical posterior pdf. A point estimate could first find a good representative of the probability mass, but then overfits to a narrow peak.

Publication I gives an example where point estimates fail completely.

2.5.2 The Laplace approximation

Compared to the point estimates, a more accurate way to approximate the integrals in Equations (2.2), (2.3), and (2.4) is to use the Laplace approximation (see Bishop, 1995; MacKay, 2003). The basic idea is to find the maximum of the function to be integrated and apply a second order Taylor series approximation for the logarithm of that function. In case of computing an expectation over the posterior distribution, the maximum is the MAP solution and the second order Taylor series corresponds to a Gaussian distribution for which integrals can be computed analytically. The Laplace approximation can be used to select the best solution in case several local maxima have been found since a broad peak is preferred over a high but narrow peak. Unfortunately the Laplace approximation does not help in situations where a good representative of the probability mass is not a local maximum, like in Figure 2.3.

Laplace approximation can be used to compare different model structures successfully. It can be further simplified by retaining only the terms that grow with the number of data samples. This is known as the Bayesian information criterion (BIC) by Schwarz (1978). Publication IX uses BIC in structural learning of logical hidden Markov models.

2.5.3 Expectation-maximisation algorithm

The expectation-maximisation (EM) algorithm (Dempster et al., 1977; Neal and Hinton, 1999; MacKay, 2003) can be seen as a hybrid of point estimates and accurate Bayesian treatment. Recall the division of unknown variables $\mathbf{\Theta}$ into parameters $\boldsymbol{\theta}$ and latent variables $\boldsymbol{S}$ in Section 2.3. EM uses point estimates for parameters $\boldsymbol{\theta}$ and distributions for latent variables $\boldsymbol{S}$. The idea is that updating is easy when these two are updated alternately. In the E-step, given certain values for parameters $\boldsymbol{\theta}$, the posterior distribution of latent variables $\boldsymbol{S}$ can be solved. In the M-step, the parameters $\boldsymbol{\theta}$ are updated to maximise likelihood or posterior density, given a fixed distribution over latent variables $\boldsymbol{S}$.

Originally the EM algorithm was used for maximum likelihood estimation in the presence of missing data. Later it was noted that latent variables can be seen as missing data and priors can be included to get the MAP estimate.

Note that EM does not only yield a posterior approximation, but also gives practical rules for iterative updates. In a sense, EM is a recipe or meta-algorithm which is used to devise particular algorithms using model-specific E and M steps.

EM is popular for its simplicity but speeding up the EM algorithm has been a topic of interest since its formulation (Meng and van Dyk, 1995). Petersen et al. (2005) analyse slowness in the limit of low noise (see also Raiko, 2001, Figure 6.1). It can be more effective to update both $\boldsymbol{S}$ and $\boldsymbol{\theta}$ at the same time by for instance using gradient-based algorithms (for comparison, see Kersting and Landwehr, 2004), hybrids of EM and gradient methods (Salakhutdinov et al., 2003), or complementing alternative updates with line search (Honkela et al., 2003). Also, there are many benefits from choosing a posterior approximation that is a distribution instead of a single set of values, such as robustness against overfitting and well-founded model selection.

The EM algorithm is adapted for parameter estimation of logical hidden Markov models in Publication VII.

2.5.4 Markov chain Monte Carlo methods

Markov Chain Monte Carlo (MCMC) methods (Gelman et al., 1995; MacKay, 2003) approximate the posterior distribution by generating a sequence of random samples from it. The integrals in Equations (2.2) and (2.4) are approximated by summing over the samples. There are many algorithms for the actual sampling, the best-known being Metropolis-Hastings algorithm and the Gibbs sampler (MacKay, 2003; Haykin, 1999). A popular method by Neal (2001) for approximating the integral in Equation (2.3) is known as annealed importance sampling.

Gibbs sampling introduced by Geman and Geman (1984) is as follows. Known variables are fixed to their values and unknown variables are given some initial values. The value of some unknown variable is updated by sampling from its conditional probability distribution assuming all the other variables fixed. This is done repeatedly for all unknown variables. First samples are discarded but the rest represent the posterior distribution, that is, expected values are estimated as sample means and so on. Assuming that all states can be reached by this process, the limiting distribution of the process is the true posterior distribution. Gibbs sampling is used for instance in the BUGS project by Spiegelhalter et al. (1995) and by Hofmann and Tresp (1996, 1998).

Sampling methods are very general and very accurate assuming enough computational resources. Unfortunately they are often computationally very demanding. Also, it is difficult to say when the process has converged (MacKay, 2003). For instance in the left subfigure of Figure 2.1, it is astronomically rare that the Gibbs sampler would find its way from one solution mode to the other.

Expectation over the samples for the parameters is often used for interpretation (including visualisation) purposes, or for fast application phase after learning. This can be problematic when the posterior distribution is multi-modal. Latent variable models often exhibit symmetries with respect to permuting the latent variables or changing signs of pairs of variables as in the left subfigure of Figure 2.1. In this case, the expected values are completely meaningless.

2.5.5 Variational approximations

Variational Bayesian (VB) methods fit a distribution of a simple form to the true posterior density. VB is sensitive to probability mass rather than to probability density. This gives it advantages over point estimates: it is robust against overfitting, and it provides a cost function suitable for learning model structures. If the true posterior has more than one mode (or cluster), VB solution finds just one of them, as shown in the left subfigure of Figure 2.1. VB provides a criterion for learning, but leaves the algorithm open. Variational Bayesian learning will be described in more detail in Section 4.1.

Depending on the form of the approximating distribution, variational Bayesian density estimates can be computationally almost as efficient as point estimates. Roberts and Everson (2001) compared Laplace approximation, sample based, and variational Bayesian learning in an independent component analysis problem on music data. The sources were well recovered using VB learning and the approach was considerably faster than the sample-based methods. Beal and Ghahramani (2003) compared VB, BIC, and annealed importance sampling in scoring model structures. VB gave a good compromise being a lot more accurate than BIC, and about a hundred times faster than sampling with comparable accuracy.

Expectation propagation by Minka (2001) is closely related to VB. A parametric distribution is fitted to the true posterior, but the measure of misfit is different. It aims at a posterior approximation that contains the whole solution. VB approximation works the other way around: the whole approximation should be contained within the solution. The difference is most apparent in cases where the posterior is multi-modal, like in the left subfigure of Figure 2.1. An approximation that contains both modes, also contains a lot of areas with low probability in between. In such cases it is reasonable to select a single mode. Expectation propagation is an algorithm whereas VB is a criterion. Unfortunately the convergence of the expectation propagation algorithm cannot be guaranteed.

Chapter 3

Graphical models

Graphical models (Pearl, 1988; Jensen et al., 1990; Cowell et al., 1999; Neapolitan, 2004; Bishop, 2006; Murphy, 2001) provide a formalism for defining the structure for a probabilistic model. A graphical model is a graph whose nodes represent random variables and edges represent direct dependencies. The models presented here vary mostly in whether they are static or dynamic and whether the variables are discrete or continuous valued.

Graphical models have evolved from being a mere academic curiosity into a popular field of research with a huge number of applications. The applications range from network engineering to bioinformatics.

All the models and methods studied in this thesis can be seen as extensions of these basic models, which are therefore introduced here.

3.1 Well-known graphical models

In the following, well-known graphical models are described. Most of them have been invented before the general framework and thus, the graphical model framework can be seen as a way to view all of these methods as instances of a common formalism. Using the shared formalism, developments in one field are easier to transfer to others (Jordan, 1999).

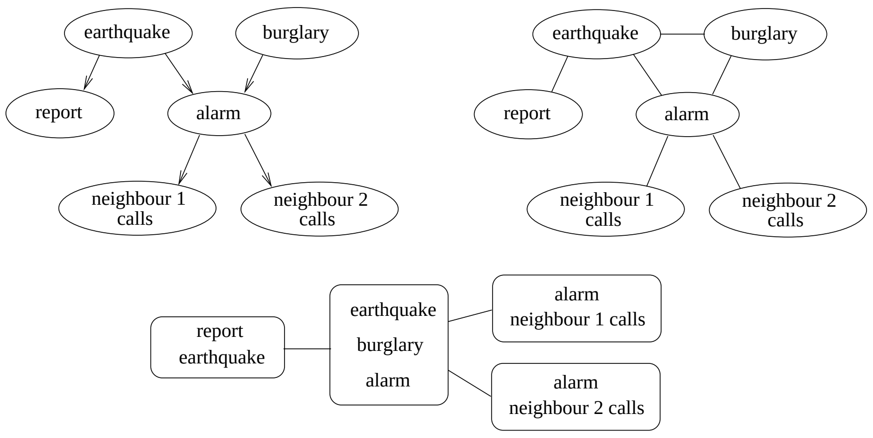


Figure 3.1: Top left: Bayesian network for the alarm example. Top right: A Markov network formed by connecting all parents that share a common child and removing edge directions. Bottom: The corresponding join tree whose nodes correspond to cliques of the Markov network.

3.1.1 Bayesian networks

Let us study an example by Pearl (1988). Mr. Holmes receives a phone call from his neighbour about an alarm in his house. As he is debating whether or not to rush home, he remembers that the alarm might have been triggered by an earthquake. If an earthquake had occurred, it might be on the news, so he turns on his radio. The events are shown in Figure 3.1. Marking the events earthquake by E, burglary by B and so on, the joint probability can be factored into

$$\begin{aligned} &P(E, B, R, A, N_1, N_2) = \\ &P(E)P(B \mid E)P(R \mid E, B)P(A \mid E, B, R)P(N_1 \mid E, B, R, A)P(N_2 \mid E, B, R, A, N_1) \\ &= P(E)P(B)P(R \mid E)P(A \mid E, B)P(N_1 \mid A)P(N_2 \mid A). \quad (3.1) \end{aligned}$$

The first equation is just basic manipulation of probabilities and the second equation represents conditional independencies. For instance, the probability of an earthquake report on the radio R does not depend on whether there is a burglary B or not, given that we know whether there was an earthquake or not, so $P(R \mid E, B) = P(R \mid E)$.

	N_1=true	N_1=false
A=true	0.7	0.3
A=false	0.001	0.999

Table 3.1: The conditional probability table describing $P(N_1 \mid A)$.

The graphical representation of the conditional probabilities depicted in the top left subfigure of Figure 3.1 is intuitive: arrows point from causes to effects. Note that the graph has to be acyclic, no occurrence can be its own cause. Loops, or cycles when disregarding directions of the edges, are allowed. Assuming that the variables are discrete, the conditional probabilities are described using a conditional probability table that lists probabilities for all combinations of values for the relevant variables. For instance, $P(N_1 \mid A)$ is described in Table 3.1. Recalling the notation from Section 2.3, the numbers in the conditional probability table are parameters $\boldsymbol{\theta}$ while unobserved variables in the nodes are $\boldsymbol{S}$.

Inference in Bayesian networks solves questions such as "what is the probability of burglary given that neighbour 1 calls about the alarm?" Approximate inference methods such as sampling (see Section 2.5.4) can be used, but in relatively simple cases, also exact inference can be done. For that purpose, we will form the *join tree* of the Bayesian network. First, we connect parents of a node to each other, and remove all directions of the edges, (see top right subfigure of Figure 3.1). This is called a Markov network. Next, we find all cliques (fully connected subgraphs) of the Markov network. Then we create a join tree[1] where nodes represent the cliques in the Markov network and edges are set between nodes who share some variables (see bottom subfigure of Figure 3.1).

We can rewrite the joint distribution in Equation 3.1 as

$$P(E, B, R, A, N_1, N_2) = \frac{P(R, E)P(E, B, A)P(A, N_1)P(A, N_2)}{P(E)P(A)P(A)}, \tag{3.2}$$

where the numerator is a product of distributions of nodes in the join tree and the denominator is the product of the distributions of the edges of the join tree. Then the inference question "what is the probability of burglary given that neighbour 1 calls about the alarm?" is solved by substituting $N_1 = true$ and marginalising

[1]We assume for now that the join tree is really a tree.

the uninteresting variables away:

$$\begin{aligned} P(B \mid N_1 = \text{true}) &= \frac{P(B, N_1 = \text{true})}{P(N_1 = \text{true})} \\ &\propto P(B, N_1 = \text{true}) \\ &= \sum_{E,R,A,N_2} P(E, B, R, A, N_1 = \text{true}, N_2) \\ &= \sum_{E,R,A,N_2} \frac{P(R, E)P(E, B, A)P(A, N_1 = \text{true})P(A, N_2)}{P(E)P(A)P(A)}, \end{aligned} \tag{3.3}$$

where $\propto$ means "proportional to". The constant $P(N_1 = \text{true})$ can be ignored if the resulting distribution is normalised in the end. Summing over all the latent variables is often prohibitively computationally expensive, so better means have been found.

For efficient marginalisation in join trees, there is an algorithm called belief propagation (Pearl, 1988) based on message passing. First, any one of the nodes in the join tree is selected as a root. Messages π are sent away from the root and messages λ towards the root. The marginal posterior probability of a node X in the join tree given observations $\mathbf{e}$ is decomposed into two parts:

$$\begin{aligned} P(X \mid \mathbf{e}) &\propto P(X \mid \mathbf{e}_{\text{anc(X)}})P(\mathbf{e}_{\text{desc(X)}} \mid X) \\ &= \pi(X)\lambda(X), \end{aligned} \tag{3.4}$$

where $\mathbf{e}_{\text{anc(X)}}$ and $\mathbf{e}_{\text{desc(X)}}$ are the observations in the ancestor and descendant nodes of X in the join tree accordingly. The messages can be computed recursively by

$$\pi(X) = \sum_Y P(X \mid Y)\pi(Y) \tag{3.5}$$

$$\lambda(Y) = \prod_j \lambda_j(Y) \tag{3.6}$$

$$\lambda_X(Y) = \sum_X \lambda(X)P(X \mid Y) \tag{3.7}$$

where Y is the parent of X in the join tree and j are its children, including X. The message π for the root node is set to the prior probability of the node, and the messages λ are set to uniform distribution for the unobserved leaf nodes. The belief propagation algorithm extended for logical hidden Markov models in Publication VII.

Returning to the example in Equation 3.2 and selecting node (E, B, A) as the root of the join tree, the necessary messages are

$$\pi(E, B, A) = P(E, B, A) \tag{3.8}$$

$$\lambda_{(R,E)}(E, B, A) = \sum_R \lambda(R) P(R \mid E, B, A) \tag{3.9}$$

$$\lambda_{(A,N_1)}(E, B, A) = \sum_{N_1} \lambda(N_1) P(N_1 \mid E, B, A) \tag{3.10}$$

$$\lambda_{(A,N_2)}(E, B, A) = \sum_{N_2} \lambda(N_2) P(N_2 \mid E, B, A). \tag{3.11}$$

Since $\lambda(R)$ and $\lambda(N_2)$ are uniform (unobserved leaf nodes), it is easy to see that $\lambda_{(R,E)}(E, B, A)$ and $\lambda_{(A,N_2)}(E, B, A)$ are also uniform and can thus be ignored. We get

$$\begin{aligned} P(B \mid N_1 = \text{true}) &= \sum_{E,A} \pi(E, B, A) \lambda(E, B, A) \\ &= \sum_{E,A} P(E, B, A) \lambda_{(A,N_1)}(E, B, A) \\ &= \sum_{E,A} P(E, B, A) P(N_1 = \text{true} \mid E, B, A) \\ &= \sum_{E,A} P(E) P(B) P(A \mid E, B) P(N_1 = \text{true} \mid A). \end{aligned} \tag{3.12}$$

For the inference to be exact, the join tree must not have loops (like the name implies). Loopy belief propagation by Murphy et al. (1999) uses belief propagation algorithm regardless of loops. Experiments show that in many cases it still works fine. The messages are initialised uniformly and iteratively updated until the process hopefully converges. It is also possible to find an exact solution by getting rid of loops at the cost of increased computational complexity. This happens by adding edges to the Markov network.

Bayesian networks can manage continuous valued variables, when three simplifying assumptions are made (Pearl, 1988). All interactions between variables are linear, the sources of uncertainty are normally distributed, and the causal network is singly connected (no two nodes share both common descendants and common ancestors). Instead of tables, the conditional probabilities are described with linear mappings. Posterior probability is Gaussian.

Bayesian networks are static, that is, each data sample is independent from others. The extension to dynamic (or temporal) Bayesian networks (see Ghahramani,

1998) is straightforward. Assume that the data samples $\mathbf{x}$, e.g. $\mathbf{x} = (E, B, R, A, N_1, N_2)$, are indexed by time t. Each variable can have as parents variables of sample $\mathbf{x}(t-1)$ in addition to the variables of the sample itself $\mathbf{x}(t)$. Hidden Markov models (see Section 3.1.5) are an important special case of dynamic Bayesian networks.

Linearity assumption can be relaxed when some approximations are made. Section 4.4 and Publication I present such a framework based on variational Bayesian learning. There are also other approaches. Hofmann and Tresp (1996) study Bayesian networks with nonlinear conditional density estimators. Inference is based on Gibbs sampling and learning is based on cross-validated maximum a posteriori estimation.

3.1.2 Markov networks

Markov networks (Pearl, 1988), historically also known as Markov random fields, are undirected graphical models. A Markov network does not commit on whether A caused B, it is interested only whether there is a dependency or not. A popular application is images where pixels are variables and edges are drawn between neighbouring pixels.

Since inference in Bayesian networks was explained by first transforming it into a Markov network, inference in Markov networks does not require much additional attention. We just start directly from the undirected graph, like in the top right subfigure of Figure 3.1. The joint density can be written directly as in Equation 3.2, but the standard way of writing the joint distribution is different. The joint distribution is proportional to the product of potentials ψ over the cliques of the network, for example:

$$P(E, B, R, A, N_1, N_2) \propto \psi(R, E)\psi(E, B, A)\psi(A, N_1)\psi(A, N_2). \tag{3.13}$$

Like Bayesian networks, Markov networks can manage continuous values with same simplifying assumptions that can be relaxed by resorting to approximations. Hofmann and Tresp (1998) introduce nonlinear Markov networks. Each continuous valued variable x_i is modelled using all of its neighbours $\mathcal{B}_i$ in the network. The modelled conditional densities $p^M(x_i \mid \mathcal{B}_i)$ can be directly used for Gibbs sampling. The complete likelihood function involves some integrals which cannot be solved in closed form but need to be approximated numerically. Taskar et al. (2002) introduce relational Markov networks (RMN) where the structure of the Markov network is defined by the relational data. Each variable might have a different number of neighbours, but generalisation is possible due to shared clique potentials.

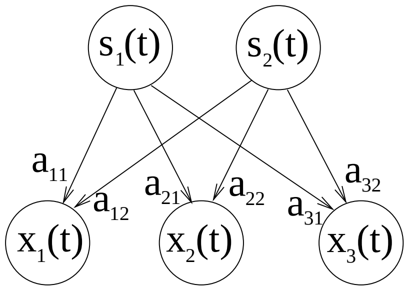


Figure 3.2: Graphical representations of factor analysis, principal component analysis, and independent component analysis are the same. Latent variables in the top layer are fully connected to the observations in the bottom layer. In this case, the vectors $\mathbf{s}(t)$ are two dimensional and $\mathbf{x}(t)$ are three dimensional.

Section 6.3 and Publication VI extend these ideas into nonlinear relational Markov networks.

3.1.3 Factor analysis and principal component analysis

Factor analysis (Harman, 1967; Kendall, 1975; Hyvärinen et al., 2001) (FA) can be seen as a Bayesian network consisting of two layers, depicted in Figure 3.2. The top layer contains latent variables $\mathbf{s}(t)$ and the bottom layer contains observations $\mathbf{x}(t)$. The two layers are fully connected, that is, each observation has all of the latent variables as its parents. The index t stands for the data case.

The mapping from factors to data is linear[2]

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t), \tag{3.14}$$

or componentwise

$$x_i(t) = \sum_j a_{ij} s_j(t) + n_j(t), \tag{3.15}$$

where $\mathbf{n}(t)$ is noise or reconstruction error vector. Typically the dimensionality of the factors is smaller than that of the data. Factors and noise are assumed to have a Gaussian distribution with an identity and diagonal covariance matrix,

[2]Note that the data is assumed to be zero mean here to simplify the equations.

respectively. Recalling the notation from Section 2.3, parameters $\boldsymbol{\theta}$ include the weight matrix $\mathbf{A}$ and noise covariance for $\mathbf{n}(t)$.

Equation (3.14) does not fix the matrix $\mathbf{A}$, since there is a group of rotations that yields identical observation distributions. Several criteria have been suggested for determining the rotation. One is parsimony, which roughly means that most of the values in $\mathbf{A}$ are close to zero. Another one leads to independent component analysis described in Section 3.1.4. Sections 4.2 and 4.4.2 describe extensions of factor analysis releasing from the linearity assumption of the dependency between factors and observations.

Principal component analysis (PCA) (Jolliffe, 1986; Kendall, 1975; Hyvärinen et al., 2001), equivalent to the Hotelling transform, the Karhunen-Loève transform, and the singular value decomposition, is a widely used method for finding the most important directions in the data in the mean-square sense. It is the solution of the FA problem under low noise (see Bishop, 2006) with orthogonal principal components (the columns of the weight matrix $\mathbf{A}$).

The first principal component $\mathbf{a}_1$ corresponds to the line on which the projection of the data has the greatest variance:

$$\mathbf{a}_1 = \arg \max_{||\boldsymbol{\xi}||=1} \sum_{t=1}^{T} (\boldsymbol{\xi}^T \mathbf{x}(t))^2. \tag{3.16}$$

The other components are found recursively by first removing the projections to the previous principal components:

$$\mathbf{a}_k = \arg \max_{||\boldsymbol{\xi}||=1} \sum_{t=1}^{T} \left[\boldsymbol{\xi}^T \left(\mathbf{x}(t) - \sum_{i=1}^{k-1} \mathbf{a}_i \mathbf{a}_i^T \mathbf{x}(t) \right) \right]^2 . \tag{3.17}$$

There are many other ways to formulate PCA, including probabilistic PCA (Bishop, 1999). In practice, the principal components are found by calculating the eigenvectors of the covariance matrix $\mathbf{C}$ of the data

$$\mathbf{C} = E \left\{ \mathbf{x}(t) \mathbf{x}(t)^T \right\} \tag{3.18}$$

The eigenvalues are positive and they correspond to the variances of the projections of data on the eigenvectors. The weight matrix $\mathbf{A}$ is formed from the eigenvectors and it is always orthogonal.

3.1.4 Independent component analysis

The mixing model of independent component analysis (ICA) is similar to that of the FA (see Figure 3.2), but in the basic case with equal number of factors (or components) as observations and without the noise term. The data is thought to have been generated from independent components $\mathbf{s}(t)$ through a square mixing matrix $\mathbf{A}$ by

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t). \tag{3.19}$$

The components $\mathbf{s}(t)$ are independent, that is,

$$p(s_1, s_2, \ldots, s_n) = p_1(s_1)p_2(s_2)\ldots p_n(s_n), \tag{3.20}$$

and the assumption of Gaussianity of the components is relaxed. Assuming that at most one of the independent components is Gaussian, the model is identifiable (Comon, 1994).

In the basic case, the number of components is the same as the number of observation and the components can be reconstructed from data given the mixing matrix $\mathbf{A}$ by $\mathbf{s}(t) = \mathbf{A}^{-1}\mathbf{x}(t)$. The independence of the reconstructed $\mathbf{s}(t)$ can then be measured for instance by the non-Gaussianity of the components or by mutual information (Hyvärinen et al., 2001). ICA can thus be done by iteratively maximising such a measure. ICA has many fields of applications, such as brain imaging (Vigário et al., 1998), telecommunications (Raju et al., 2006; Ristaniemi, 2000), speech separation, and so on (Hyvärinen et al., 2001).

ICA can be approached from different starting points. It can be viewed as a Bayesian network when the one dimensional distributions for the components are modelled with for example mixtures-of-Gaussians (Attias, 1999, 2001; Choudrey et al., 2000; Miskin and MacKay, 2001). This is also known as independent factor analysis. Extensions to the basic ICA involve additive noise, convolutive or nonlinear mixing, and the number of components might differ from the number of observations (Hyvärinen et al., 2001).

Publication III studies the reconstruction of corrupted values in data by independent factor analysis.

3.1.5 Hidden Markov models

Hidden Markov models (HMMs) (Rabiner and Juang, 1986) are dynamic Bayesian networks with a certain structure. Observed data are assumed to be generated as a function of an unobserved state which is evolving stochastically in time. The

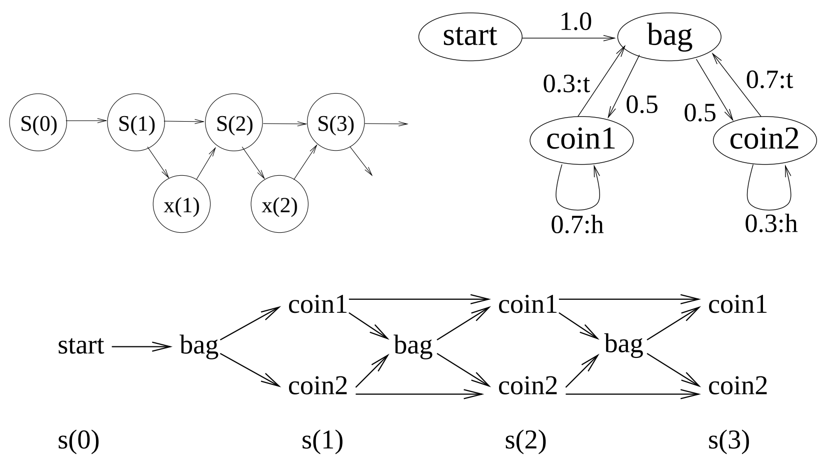


Figure 3.3: Graphical representations of a hidden Markov model. *Top left:* The corresponding Bayesian network. *Top right:* The transitions are represented with an arrow and annotated with transition probabilities and possible observations. *Bottom:* The trellis showing all the possible evolutions of states unrolled in time.

observations $x(t)$ are conditioned on state $s(t)$. The discrete state $s(t)$ at time t is a latent variable and the conditional probability for a state transition $P(s(t) \mid s(t-1))$ does not depend on t. Here we will use a non-standard formulation where the state transition is conditioned on the observation as well, that is, $P(s(t) \mid s(t-1), x(t-1))$. It is fairly easy to see that the two approaches are equivalent[3] in representational capacity. Appendix B of Publication VII shows that the two approaches are equivalent in a setting generalised to first-order logic.

Figure 3.3 shows an example of a HMM. There are two bent coins in a bag, one gives more heads and the other more tails. A person draws a coin randomly from the bag and tosses it until it lays tails up. At that point it is put back to the bag and one of the coins is drawn and tossed again. At time 0, the system is always in the *start* state ($s(0)$ =start). Afterwards, it is always either in state *coin1* or *coin2* possibly going through an auxiliary state *bag*. The state being hidden (or latent) means that one does not know which coin is used, only the sequence of heads and tails is observed.

[3]By either ignoring the conditioned $x(t-1)$ or including a copy of the observation to the hidden state.

The instance of belief propagation algorithm for HMMs is called the forward-backward algorithm. In the example, this would give the probabilities for which coin was drawn from the bag at each stage. It is also possible to learn the parameters from observations, say, estimate how badly the coins are bent. That can be done using the Baum-Welch algorithm, which is the instance of EM algorithm for HMMs.

Hidden Markov models are very popular for analysing sequential data. Application areas include computational biology (Koski, 2001), speech recognition (Rabiner and Juang, 1986), natural language processing (Charniak, 1993), user modelling (Lane, 1999), and robotics (Meila and Jordan, 1996).

There are several extensions of HMMs such as factorial HMMs by Ghahramani and Jordan (1997), relational Markov models by Anderson et al. (2002), and extension based on tree automata by Frasconi et al. (2002). Section 6.2 and Publication VII introduce an extension of hidden Markov models to first-order logic, which generalises all of the above.

3.1.6 State-space models

Linear state-space models (see textbook by Chen, 1999) share the same structure as hidden Markov models but now the states $\mathbf{s}(t)$ and observations $\mathbf{x}(t)$ are continuous valued vectors. The conditional probabilities are represented as a linear mapping with additive Gaussian noise:

$$\mathbf{s}(t) = \mathbf{B}\mathbf{s}(t-1) + \mathbf{n}_s(t) \tag{3.21}$$

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}_x(t) \tag{3.22}$$

Note that Equation (3.22) is exactly the same as (3.14), that is, state-space models can be seen as a dynamic extension of factor analysis. The dynamics in Equation (3.21) correspond to linear dynamical systems discretised in the time domain.

Again, the belief propagation algorithm has another name, and it was originally derived from a different starting point probably by a Danish statistician T.N. Thiele in 1880 and later popularised by Kalman (1960) (see also the textbook by Anderson and Moore, 1979). In the context of state-space models, belief propagation is known as the Kalman smoother. It is popular in many applications fields, including econometrics (Engle and Watson, 1987), radar tracking (Chui and Chen, 1991), control systems (Maybeck, 1979), signal processing, navigation, and robotics.

In control systems, dynamics can be affected by control inputs $\mathbf{u}(t)$. Equation

(3.21) for dynamics is replaced by

$$\mathbf{s}(t) = \mathbf{B}\mathbf{s}(t-1) + \mathbf{C}\mathbf{u}(t) + \mathbf{n}_s(t). \tag{3.23}$$

Note that the control inputs $\mathbf{u}(t)$ are coming from outside the generative model. One possibility is to use feedback control (see textbook by Doyle et al., 1992),

$$\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t-1) + \mathbf{r}(t), \tag{3.24}$$

but note that $\mathbf{K}$ and $\mathbf{r}(t)$ should be chosen so as to accomplish the control goal, not by inference or learning.

Section 4.3 studies an extension to nonlinear state-space models where the linear mappings $\mathbf{A}$ and $\mathbf{B}$ are replaced by multi-layer perceptron networks. Section 4.3.2 and Publication IV study control in nonlinear state-space models.

3.2 Tasks

This thesis studies four tasks that can be done with graphical models: inferring the distributions for latent variables, estimating (or learning) parameters of the model, learning the structure of the model, and making decisions. Each of them is described in turn.

3.2.1 Inference

Inference is the task of computing the posterior probability over the latent variables $\boldsymbol{S}$ given a fixed set of parameters $\boldsymbol{\theta}$, the data $\boldsymbol{X}$ and the model structure $\mathcal{H}$, according to the Bayes rule (Equation 2.1). The distribution is often very high dimensional and for all practical purposes it is represented as marginal distributions (see Eq. 2.2) over groups of variables. The computations are not straightforward and therefore one needs to use algorithms such as belief propagation, described in Section 3.1.1.

One of the advantages of graphical models is that handling of missing values in data is straightforward and consistent. Instead of belonging to data $\boldsymbol{X}$, missing values belong to latent variables $\boldsymbol{S}$ and their reconstructions (or posterior distributions) are inferred as any other latent variables. Reconstruction of missing values in linear and nonlinear models is studied in Section 4.1.4 and Publication II.

Exact inference by belief propagation has exponential computational complexity with respect to the size of the largest clique in the Markov network (see Figure 3.1), so often one needs to settle for approximated inference. In some extensions such as nonlinear state-space models described in Section 4.3, there is no analytical solution at all. Different kinds of approximate methods are described in Section 2.5.

3.2.2 Parameter learning

The task of learning the parameters of a model means that given a set of data cases or observations $\boldsymbol{X}$ and a model structure $\mathcal{H}$, one can infer the distribution over the model parameters $\boldsymbol{\theta}$, found for instance in the conditional probability table in Table 3.1, in the clique potentials ψ in Equation (3.13), in the mapping $\mathbf{A}$ in Equation (3.14), or the transition probabilities in Figure 3.3. Parameter learning does not differ from inference in Bayesian probability theory, so the reasons for studying them separately are mostly practical. For instance in the EM algorithm, the updates of parameters and latent variables are done separately and in different ways. Also, local update rules work efficiently in parameter learning, whereas explicit propagation of information is important in state inference of dynamic systems (see Publication V).

Parameter learning can be really simple. Consider learning the values in the conditional probability table for neighbour 1 calling about the alarm in case there is an alarm or not, $P(N_1 \mid A)$ in Table 3.1. Given data samples where we observe whether there was an alarm and whether the neighbour called or not, let us settle for a point estimate: the most likely set of parameters. The ML solution is simply to count how many times each of the four cases appear in the data and turn them into probabilities by normalising each row to sum to one.

3.2.3 Structural learning

Also the structure of the model can be learned from data. This includes adding edges between nodes and possibly new nodes. A straightforward way of learning the structure is to try out different structures $\mathcal{H}$ and select the one with the largest marginal likelihood $P(\boldsymbol{X} \mid \mathcal{H})$ (see Equation 2.3). Depending on the approximations, sometimes more complex models need to be explicitly penalised (MacKay, 1995b).

Just like parameter learning includes inference as a subproblem, structural learning includes parameter learning as a subproblem. To measure the model evidence for a structure, parameters need to be learned. A standard way of searching is to generate candidate structures by making minimal changes in the current hypothesis, such as adding, deleting, or reversing an edge. Parameters of the current hypothesis can be used as a good first guess for the parameters of the candidates. In a greedy search, the best candidate is selected as the next hypothesis. Getting stuck in a local minimum can be avoided for instance by using simulated annealing by Kirkpatrick et al. (1983) (Haykin, 1999).

Structural EM by Friedman (1998) speeds up structural search. The posterior distribution over the latent variables is inferred for the current hypothesis and fixed. Then, candidate hypotheses are evaluated using this distribution, that is, only the parameters are updated. As before, one of the candidates is selected as the next current hypothesis and the inference is done again. Now, instead of running EM algorithm for each candidate structure, only a single M-step of the EM algorithm is needed. Candidate evaluation is not as accurate as before, but this is easily compensated by the large speed-up. The structural learning algorithm for logical hidden Markov models, introduced in Publication IX, is based on generalised structural EM.

There is a second broad class of algorithms for structural learning besides performing a search. These are known as independence-based or constraint-based algorithms. For example, Bromberg et al. (2006) discover structures of Markov networks using independence tests.

3.2.4 Decision making

The fourth task, decision making, differs a lot from the other three. The task is to select actions that maximise expected utility, as explained in Section 2.4. The actions or controls appear as external inputs in the graphical model, such as the control inputs $\mathbf{u}(t)$ in Section 3.1.6.

Utility, like random variables, can also be decomposed into nodes. The global utility is a sum of local utilities. A utility node has as parents all the actions and random variables on which it depends. Now, the values for action nodes can be selected to maximise expected utility (Cowell et al., 1999). The resulting graph is called an influence diagram (Pearl, 1988).

In model-predictive control (e.g. Eduardo Fernández Camacho, 2004), actions can be selected as follows. First, some initial guess for the actions is made. Latent variables and utilities are inferred for a given sequence of actions. The gradient of total expected utility with respect to each random variable and action is propagated backwards to each action. Actions are updated in the direction of increasing utility, and the process is iterated. Application of this to control in nonlinear state-space models is described in Section 4.3.2 and Publication IV.

Chapter 4

Variational learning of nonlinear graphical models

This chapter describes extensions of graphical models with continuous values to the case where the linearity assumption is relaxed. These are known as nonlinear graphical models. For reasons explained in Section 2.5, nonlinear graphical model suit well to be handled using variational Bayesian methods.

Section 4.1 describes variational Bayesian methods in general and the rest of the chapter describes some particular models.

4.1 Variational Bayesian methods

Variational Bayesian (VB) learning (Barber and Bishop, 1998; Hinton and van Camp, 1993; Lappalainen and Miskin, 2000; MacKay, 1995a, 2003; Jordan et al., 1999; Lappalainen and Honkela, 2000) is a fairly recently introduced (Wallace, 1990; Hinton and van Camp, 1993) approximate fully Bayesian method, which has become popular because of its good properties. Its key idea is to approximate the exact posterior distribution $p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H})$ by another distribution $q(\boldsymbol{\Theta})$ that is computationally easier to handle.

Typically, the misfit of the approximation is measured by the Kullback-Leibler (KL) divergence between two probability distributions $q(v)$ and $p(v)$. The KL divergence is defined by

$$D(q(v) \parallel p(v)) = \int q(v) \ln \frac{q(v)}{p(v)} dv \geq 0 \tag{4.1}$$

which measures the difference in the probability mass between the densities $q(v)$ and $p(v)$. Its minimum value 0 is achieved when the densities $q(v)$ and $p(v)$ are the same.

The VB method works by iteratively minimising the misfit between the actual posterior pdf and its parametric approximation using the KL divergence. Note that VB learning defines the goal and a performance measure, but leaves the actual algorithm open. The approximating distribution $q(\boldsymbol{\Theta})$ is usually chosen to be a product of several independent distributions, one for each parameter or a set of similar parameters. Even a crude approximation of a diagonal multivariate Gaussian density is adequate for finding the region where the mass of the actual posterior density is concentrated. The mean values of the Gaussian approximation provide reasonably good point estimates of the unknown parameters, and the respective variances measure the reliability of these estimates. An example is given in Figure 2.1.

A main motivation of using VB is that it avoids overfitting which would be a difficult problem if ML or MAP estimates were used (see Section 2.5). VB method allows one to select a model having appropriate complexity, making often possible to infer the correct number of sources or latent variables.

Variational Bayes is closely related to information theoretic approaches which minimise the description length of the data, because the description length is defined to be the negative logarithm of the probability. Minimal description length thus means maximal probability. The information theoretic view provides insights to many aspects of learning and helps explain several common problems (Honkela and Valpola, 2004; Hinton and van Camp, 1993).

4.1.1 Cost function

The basic idea in variational Bayesian learning is to minimise the misfit between the exact posterior pdf $p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H})$ and its parametric approximation $q(\boldsymbol{\Theta})$. The misfit is measured here with the Kullback-Leibler (KL) divergence

$$\begin{aligned}\mathcal{C}_{KL} = D(q(\boldsymbol{\Theta}) \parallel p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H})) &= \left\langle \ln \frac{q(\boldsymbol{\Theta})}{p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H})} \right\rangle \\ &= \int q(\boldsymbol{\Theta}) \ln \frac{q(\boldsymbol{\Theta})}{p(\boldsymbol{\Theta} \mid \boldsymbol{X}, \mathcal{H})} d\boldsymbol{\Theta},\end{aligned} \tag{4.2}$$

where the operator $\langle \cdot \rangle$ denotes an expectation over the distribution $q(\boldsymbol{\Theta})$. The marginal likelihood $p(\boldsymbol{X} \mid \mathcal{H})$ is hard to evaluate and therefore the cost function $\mathcal{C}$ that is actually used is

$$\mathcal{C} = \left\langle \ln \frac{q(\boldsymbol{\Theta})}{p(\boldsymbol{X}, \boldsymbol{\Theta} \mid \mathcal{H})} \right\rangle = \mathcal{C}_{KL} - \ln p(\boldsymbol{X} \mid \mathcal{H}). \tag{4.3}$$

A typical choice of posterior approximations $q(\boldsymbol{\Theta})$ is Gaussian with limited covariance matrix, that is, all or most of the off-diagonal elements are fixed to zero. Often the posterior approximation is assumed to be a product of independent factors. The factorial approximation, combined with the factorisation of the joint probability like in Equation (3.1), leads to the division of the cost function in Equation (4.3) into a sum of simple terms, and thus to a relatively low computational complexity.

Miskin and MacKay (2001) used VB learning for ICA (See Section 3.1.4). They compared two approximations of the posterior: The first was a Gaussian with full covariance matrix, and the second was a Gaussian with a diagonal covariance matrix. They noticed that the factorial approximation is computationally more efficient and still gives a bound on the evidence and does not suffer from overfitting. On the other hand, Ilin and Valpola (2005) showed that the factorial approximation favours a solution that has an orthogonal mixing matrix, which can deteriorate the performance.

4.1.2 Model selection

VB learning offers another important benefit. Comparison of different models is straightforward. The Bayes rule can be applied again to get the probability of a model given the data

$$p(\mathcal{H}_i \mid \boldsymbol{X}) = \frac{p(\boldsymbol{X} \mid \mathcal{H}_i) p(\mathcal{H}_i)}{p(\boldsymbol{X})}, \tag{4.4}$$

where $p(\mathcal{H}_i)$ is the prior probability of the model $\mathcal{H}_i$ and $p(\boldsymbol{X})$ is a constant that can be ignored. A lower bound on the evidence term $p(\boldsymbol{X} \mid \mathcal{H}_i)$ is obtained from

Equation (4.3) and it is

$$p(\boldsymbol{X} \mid \mathcal{H}_i) = \exp(\mathcal{C}_{KL} - \mathcal{C}) \geq \exp(-\mathcal{C}). \tag{4.5}$$

Multiple models can be used as a mixture-of-experts model (Haykin, 1999). The experts can be weighted with their probabilities $p(\mathcal{H}_i \mid \boldsymbol{X})$ given in equation (4.4). Lappalainen and Miskin (2000) show that the optimal weights in the sense of variational Bayesian approximation are in fact $p(\mathcal{H}_i)\exp(-\mathcal{C})$. If the models have equal prior probabilities $p(\mathcal{H}_i)$, the weights simplify further to $\exp(-\mathcal{C})$. In practice, the costs $\mathcal{C}$ tend to differ in the order of hundreds or thousands, which makes the model with the lowest cost $\mathcal{C}$ dominant. Therefore it is reasonable to concentrate on model selection rather than weighting.

4.1.3 Optimisation and local minima

Using nonlinear models leads to an optimisation problem with many local minima. This makes the method sensitive to initialisation. Typically initialisation is based on linear PCA (see Section 3.1.3). This can lead to suboptimal results if the mixing is strongly nonlinear. Honkela et al. (2004) significantly improve performance by using a nonlinear model (kernel PCA) for initialisation instead.[1]

Learning and inference are based on minimising the cost function in Equation (4.3) by iterative updates. There are two essentially different approaches for that. In the first approach, updates are local, that is, only some variables are updated while assuming that the posterior distribution over other variables stays constant. The second option is to update all variables at once. Benefits of local updating include biological motivation (all interaction in brains is local), modularity, parallelisability, and easily guaranteed convergence. Global updates, on the other hand, are often faster. Both approaches are used in this work. Honkela et al. (2003) show how local updates can be transformed into global ones.

Some parts of a latent variable model might be effectively turned off during learning. This happens when a latent variable has no effect on any of the other latent variables or observations. Such a set of parameter values is a local minimum of the cost function. In such cases, it is reasonable to either change the model structure accordingly, or reinitialise those parts. Publication I discusses these issues and measures against suboptimal local minima in detail.

[1]The use of nonlinear models is nontrivial, like choosing a good kernel in kernel PCA.

4.1.4 Missing values

Handling missing values in data is an important point in statistical analysis (Little and D.B.Rubin, 1987). Generative models can usually easily deal with missing observations, and can also be used to fill in the missing values. In supervised learning, the data is split into two parts: inputs and desired outputs. Learning data includes both, but in the end, the model is used for predicting outputs based only on the test inputs. By ignoring the splitting and creating a model for the whole data, unsupervised learning can be used for a similar task as supervised learning. Both the inputs and desired outputs of the learning data are treated equally. When a generative model for the combined data is learned, it can be used to reconstruct the missing outputs for the test data. The scheme used in unsupervised learning is more flexible because any part of the data can act as the cue which is used to complete the rest of the data. In supervised learning, the inputs always act as the cue.

The quality of the reconstructions provides insight to the properties of different unsupervised models. Self-organising maps by Kohonen (2001), factor analysis, and its nonlinear extensions were studied in Publication II by reconstructing the missing values of various data sets. Experiments were conducted using four different scenarios for the missing values. This way, different aspects of the algorithms could be studied. These included accuracy in high-dimensional data, high nonlinearity, memorisation, and generalisation. The performance of several models varied a lot according to the different settings.

One of the experiments in both Publications II and V involves missing values in speech spectrograms. Spectrograms represent energy of the frequency content in a short time window for a number of time points and frequencies. Speech spectrogram is a standard representation in speech recognition. Palomäki et al. (2004) apply the missing-data framework to recognise reverberant speech. The algorithm seeks strong speech onsets not contaminated by reverberation and speech recognition is based on only the values that are observed. This approach increases the recognition accuracy substantially.

4.1.5 Partially observed values

A single value in the data can be somewhere between being observed and missing. So-called coarse data means that we only know that a data point belongs to a certain subset of all possibilities. So-called soft or fuzzy data generalises this further by giving weights to the possibilities. Pearl (1988) handles the issue with

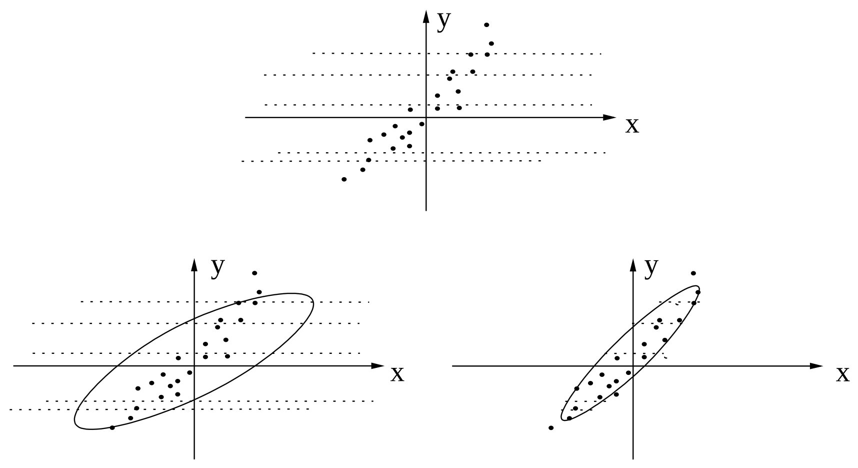


Figure 4.1: Some x-values of the data are observed only partially. They are marked with dotted lines representing their confidence intervals. *Top:* A simple data set for a factor analysis problem. *Bottom left:* In a compared approach, where a distribution is fixed over the values, the model (a Gaussian shown as the ellipse) needs to adjust to cover the distributions. *Bottom right:* In the virtual evidence approach, the partially observed values are reconstructed based on the model.

so called virtual evidence. The partially observed node itself is set as missing, but a new node is added as a child for it. Observing the new node produces a wanted likelihood term for the partially observed node.

In Publication III, different ways of handling partially observed values are studied in context of variational Bayesian learning. A simple example comparing two different approaches is given in Figure 4.1. The virtual evidence approach is recommended based on both theory and experimentation in which independent factor analysis (see Section 3.1.4) was applied to real image data.

Green et al. (2001) and Seltzer et al. (2004) study missing and unreliable values framework to improve speech recognition in noisy environment. The recognition accuracy is greatly improved in noisy environments by first identifying components in the spectrographic representation that are corrupt.

4.2 Nonlinear factor analysis

Recall factor analysis described in Section 3.1.3, in which the conditional density in Equation (3.14) is restricted to be linear. In nonlinear FA, the generative mapping from factors (or latent variables or sources) to data is no longer restricted to be linear. The general form of the model is

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t), \boldsymbol{\theta}_f) + \mathbf{n}(t)\,. \tag{4.6}$$

This can be viewed as a model about how the observations were generated from the sources. The vectors $\mathbf{x}(t)$ are observations at time t, $\mathbf{s}(t)$ are the sources, and $\mathbf{n}(t)$ are noise. The function $\mathbf{f}(\cdot)$ is a mapping from source space to observation space parametrised by $\boldsymbol{\theta}_f$.

Lappalainen and Honkela (2000) use a multi-layer perceptron (MLP) network (see Haykin, 1999) with tanh-nonlinearities to model the mapping $\mathbf{f}$:

$$\mathbf{f}(\mathbf{s}; \mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}) = \mathbf{B} \tanh(\mathbf{A}\mathbf{s} + \mathbf{a}) + \mathbf{b}\,, \tag{4.7}$$

where the tanh nonlinearity operates on each component of the input vector separately. The mapping $\mathbf{f}$ is thus parameterised by the matrices $\mathbf{A}$ and $\mathbf{B}$ and bias vectors $\mathbf{a}$ and $\mathbf{b}$. MLP networks are well suited for nonlinear FA. First, they are universal function approximators (see Hornik et al., 1989, for proof) which means that any type of nonlinearity can be modelled by them in principle. Second, it is easy to model smooth, nearly linear mappings with them. This makes it possible to learn high dimensional nonlinear representations in practice.

The traditional use of MLP networks differs a lot from the use in nonlinear FA. Traditionally MLP networks are used in a supervised manner, mapping known inputs $\mathbf{s}(t)$ to desired outputs $\mathbf{x}(t)$. During training of the network, both $\mathbf{s}(t)$ and $\mathbf{x}(t)$ are observed, whereas in nonlinear FA, $\mathbf{s}(t)$ is always latent. The traditional learning problem is much easier and can be reasonably solved by using just point estimates.

The used posterior approximation is a fully factorial Gaussian:

$$q(\boldsymbol{\Theta}) = \prod_i q(\Theta_i) = \prod_i \mathcal{N}\left(\Theta_i; \overline{\Theta}_i, \widetilde{\Theta}_i\right), \tag{4.8}$$

where the unknown variables Θ_i include the factors $\mathbf{s}$, the matrices $\mathbf{A}$ and $\mathbf{B}$, and other parameters. Thus for each unknown variable θ_i, there are two parameters, the posterior mean $\overline{\Theta}_i$ and the posterior variance $\widetilde{\Theta}_i$. The distribution that propagates through the nonlinear mapping $\mathbf{f}$ has to be approximated. Honkela and Valpola (2005) suggest to do this by linearising the tanh-nonlinearities using a Gauss-Hermite quadrature. This works better than a Taylor approximation or using a Gauss-Hermite quadrature on the whole mapping $\mathbf{f}$.

Using linear independent component analysis (ICA, see Section 3.1.4) on sources $\mathbf{s}(t)$ found by nonlinear factor analysis is a solution to the nonlinear ICA problem, that is, finding independent components that have been nonlinearly mixed to form the observations. A variety of approaches for nonlinear ICA are reviewed by Jutten and Karhunen (2004). Often, a special case known as post-nonlinear ICA is considered. In post-nonlinear ICA, the sources are linearly mixed with the mapping $\mathbf{A}$ followed by component-wise nonlinear functions:

$$\mathbf{f}(\mathbf{s}; \boldsymbol{\theta}_f) = \boldsymbol{\phi}(\mathbf{A}\mathbf{s} + \mathbf{a}), \tag{4.9}$$

where the nonlinearity $\boldsymbol{\phi}$ again operates on each element of its argument vector separately. Ilin and Honkela (2004) consider post-nonlinear ICA by variational Bayesian learning.

4.3 Nonlinear state-space models

In many cases, measurements originate from a dynamical system and form time series. In such cases, it is often useful to model the dynamics in addition to the instantaneous observations. Valpola and Karhunen (2002) extend the nonlinear factor analysis model by adding a nonlinear model for the dynamics of the sources $\mathbf{s}(t)$. This results in a state-space model where the sources can be interpreted as the internal state of the underlying generative process. On the other hand, nonlinear state-space models are a direct extension of linear state-space models (see Section 3.1.6) where the linearity assumption is relaxed.

The nonlinear static model of Equation (4.6) is extended by adding another nonlinear mapping $\mathbf{g}$ to model the dynamics. This leads to source model

$$\mathbf{s}(t) = \mathbf{s}(t-1) + \mathbf{g}(\mathbf{s}(t-1), \boldsymbol{\theta}_g) + \mathbf{n}_s(t)\,, \tag{4.10}$$

$$\mathbf{g}(\mathbf{s}; \mathbf{C}, \mathbf{D}, \mathbf{c}, \mathbf{d}) = \mathbf{D}\tanh(\mathbf{C}\mathbf{s} + \mathbf{c}) + \mathbf{d}\,, \tag{4.11}$$

where $\mathbf{s}(t)$ are the sources (states), $\mathbf{n}_s(t)$ is the Gaussian noise, and the dynamics mapping $\mathbf{g}(\cdot)$ is modelled by an MLP network.

In case the dynamic system is changing slowly, there are high correlations between consecutive states. This is taken into account by giving up the fully factorial posterior approximation used in nonlinear FA. The posterior distribution of each component i of the state vector $\mathbf{s}(t)$ is conditioned on the same component i of the state vector $\mathbf{s}(t-1)$. The approximate density $q(s_i(t) \mid s_i(t-1))$ is parameterised by the mean, linear dependence, and variance (see Valpola and Karhunen, 2002, for details).

Considering the sequence of consecutive mappings $\mathbf{g}$ in the system dynamics, where each mapping $\mathbf{g}$ consists of a linear mapping, component-wise nonlinearities, and a second linear mapping, one might think that one of the two linear mappings before and after the states is redundant since two consecutive linear mappings can always be combined into one. The second mapping allows the model to select a representation where the variational approximation is most accurate. It also allows the dimensionality of the state-space to be different from the number of used nonlinearities, thus decreasing computational complexity in some cases.

An important advantage of the VB method is its ability to learn a high-dimensional latent source space. Computational and over-fitting problems have been major obstacles in developing this kind of unsupervised methods thus far. Potential applications for the method include prediction and process monitoring, control, and speech enhancement for recognition. Is process monitoring, Ilin et al. (2004) show that VB learning is able to find a model which is capable of detecting an abrupt change in the underlying dynamics of a fairly complex nonlinear process.

4.3.1 State inference

In linear state space models, the sequence of states or sources $\mathbf{s}(1), \dots, \mathbf{s}(T)$ can be exactly inferred from data with an algorithm called the Kalman smoothing by Kalman (1960) (see also Anderson and Moore, 1979). Ghahramani and Beal (2001) show how belief propagation and the junction tree algorithms can be used in the inference in the variational Bayesian setting. As an example they perform inference in linear state-space models. Exact inference is accomplished using a single forward and backward sweep. Unfortunately these results do not apply to nonlinear state space models.

The idea behind iterated extended Kalman smoother (see Anderson and Moore, 1979) is to linearise the mappings $\mathbf{f}$ and $\mathbf{g}$ around the current state estimates $\overline{\mathbf{s}}(t)$ using the first terms of the Taylor series expansion. The algorithm alternates between updating the states by Kalman smoothing and renewing the linearisation. When the system is highly nonlinear or the initial estimate is poor, the iterated extended Kalman smoother may diverge. The iterative unscented Kalman smoother by Julier and Uhlmann (1997) replaces the local linearisation by a deterministic sampling technique. The sampled points are propagated through the nonlinearities, and a Gaussian distribution is fitted to them. The use of non-local information improves convergence and accuracy at the cost of doubling the computational complexity, but still there is no guarantee of convergence.

Particle filtering (Doucet et al., 2001) is an increasingly popular method for state inference. It generates random samples from the posterior distribution. The basic version requires a large number of particles or samples to provide a reasonable accuracy. If the state space is high dimensional, the sufficient number of samples can become prohibitively large. There are many improvements for the basic algorithm to improve efficiency. One of them, Rao-Blackwellisation (see e.g. Ristic et al., 2004), uses analytical solutions to some of the filtering equations instead of pure sampling.

Variational Bayesian inference in nonlinear state-space models is based on updating the posterior approximation of states for minimising the cost function $\mathcal{C}$. Recall that $\mathcal{C}$ is a sum of simple terms. Terms that involve a certain state $\mathbf{s}(t)$ at time t are independent of all the other states except the closest neighbours $\mathbf{s}(t-1)$ and $\mathbf{s}(t+1)$. Most optimisation algorithms would thus only consider information from the closest neighbours for each update. Information spreads around slowly because the states of different time slices affect each other only between updates. It is possible to predict this interaction by a suitable approximation.

Publication V introduces an update algorithm for the posterior mean of the states $\overline{\mathbf{s}}(t)$ by approximating total derivatives

$$\frac{d\mathcal{C}}{d\overline{\mathbf{s}}(t)} = \sum_{\tau=1}^{T} \frac{\partial \mathcal{C}}{\partial \overline{\mathbf{s}}(\tau)} \frac{\partial \overline{\mathbf{s}}(\tau)}{\partial \overline{\mathbf{s}}(t)}. \tag{4.12}$$

Once we can approximate $\frac{\partial \overline{\mathbf{s}}(t)}{\partial \overline{\mathbf{s}}(t-1)}$ and $\frac{\partial \overline{\mathbf{s}}(t)}{\partial \overline{\mathbf{s}}(t+1)}$ by linearising the mappings $\mathbf{f}$ and $\mathbf{g}$, the total derivatives are computed efficiently using the chain rule and dynamic programming. To summarise, the novel algorithm is based on minimising a variational Bayesian cost function and the novelty is in propagating the gradient $\frac{\partial \mathcal{C}}{\partial \overline{\mathbf{s}}(\tau)}$ through the state sequence.

When an algorithm is based on minimising a cost function, it is fairly easy to guarantee convergence. While the Kalman filter is clearly the best choice for inference in linear Gaussian models, the problem with many of the nonlinear generalisation is that they cannot guarantee convergence. Even when the algorithms converge, convergence can be slow. Another recent fix for convergence by Psiaki (2005) comes with a large computational cost.

Publication V compares the proposed algorithm to some of the existing methods using two experimental setups: Simulated double inverted pendulum and real-world speech spectra. The results were better than any of the comparison methods in all cases. The comparison to particle filtering was not conclusive because the particle filter was not Rao-Blackwellisised.

4.3.2 Control

Model predictive control (see Morari and Lee, 1999, for a survey) aims at controlling a dynamical system by using a predictive model. Control inputs $\mathbf{u}(t)$ are added to the nonlinear state-space model. In publication IV this is done by modifying the system dynamics in Equation (4.10) by

$$\begin{bmatrix} \mathbf{u}(t) \\ \mathbf{s}(t) \end{bmatrix} = \mathbf{g}\left(\begin{bmatrix} \mathbf{u}(t-1) \\ \mathbf{s}(t-1) \end{bmatrix}, \boldsymbol{\theta}_\mathbf{g} \right) + \mathbf{m}(t). \tag{4.13}$$

Compared to Equation (3.23), the control signals $\mathbf{u}(t)$ are not coming from outside the model, but they are modelled as well. Whereas feedback control in Equation (3.24) models control inputs as a fixed function of the observations, Equation (4.13) only gives a distribution for the control inputs and leaves the exact selection open.

Publication IV studies three different control schemes in this setting. Direct control is based on using the internal forward model directly by selecting the mean of the probability distribution given by Equation (4.13). Direct control is fast to use, but the learning of the mapping $\mathbf{g}$ is hard to do well.

The second control scheme is nonlinear model-predictive control (see e.g. Eduardo Fernández Camacho, 2004), which is based on optimising control signals based on maximising a utility function. First, an initial guess for the control signals $\mathbf{u}(t)$ is made. The posterior distribution of the future states are inferred. The gradient of total expected utility with respect to states and control signals is propagated backwards in time. Control signals are then updated in the direction of increasing utility. This process is iterated as long as there is time before the next control signal needs to be selected. Nonlinear model-predictive control can be seen as applying decision theory (see Sections 2.4 and 3.2.4).

Optimistic inference control, introduced in Publication IV, is the third studied control scheme. It is based on Bayesian inference answering the question: "Assuming success in the end, what will happen in near future?" Control signal is inferred given the history of observations and assuming wanted observations after a gap of missing values. Inference combines the internal forward model with the evidence propagating backwards from the desired future. Optimistic inference control lacks in flexibility and theoretical foundation compared to nonlinear

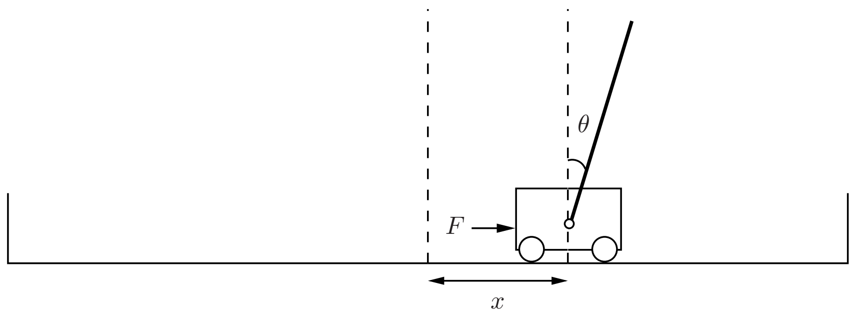


Figure 4.2: The cart-pole system. The goal is to swing the pole to an upward position and stabilise it without hitting the walls. The cart can be controlled by applying a force to it.

model-predictive control, but it provides a link between two problems: inference and control. It gave the inspiration for the inference algorithm introduced in Publication V. Tornio and Raiko (2006) apply the algorithm back in control. Attias (2003) independently discovered the idea behind optimistic inference control, calling it planning by probabilistic inference. His example, finding a goal in a grid world, is quite different from control, but the underlying idea is still the same.

The proposed control methods were tested with a cart-pole swing-up task in Figure 4.2. Figure 4.3 illustrates the model predictive control in action. The experimental results in Publication IV confirm that selecting actions based on a state-space model instead of the observation directly has many benefits: First, it is more resistant to noise because it implicitly involves filtering. Second, the observations (without history) do not always carry enough information about the system state. Third, when nonlinear dynamics are modelled by a function approximator such as an MLP network, a state-space model can find such a representation of the state that it is more suitable for the approximation and thus more predictable.

Model development is by far the most critical and time-consuming step in implementing a model predictive controller (Morari and Lee, 1999). The analysis in Publication IV is of course very shallow compared to the huge mass of control literature but there seems to be need for sophisticated model learners (or system identifiers). For instance, Rosenqvist and Karlström (2005) also learn a nonlinear state-space model for control. The nonlinearities are modelled using piecewise linear mappings. Parameters are estimated using the prediction error method, which is equivalent to the maximum likelihood estimation in the Bayesian framework.

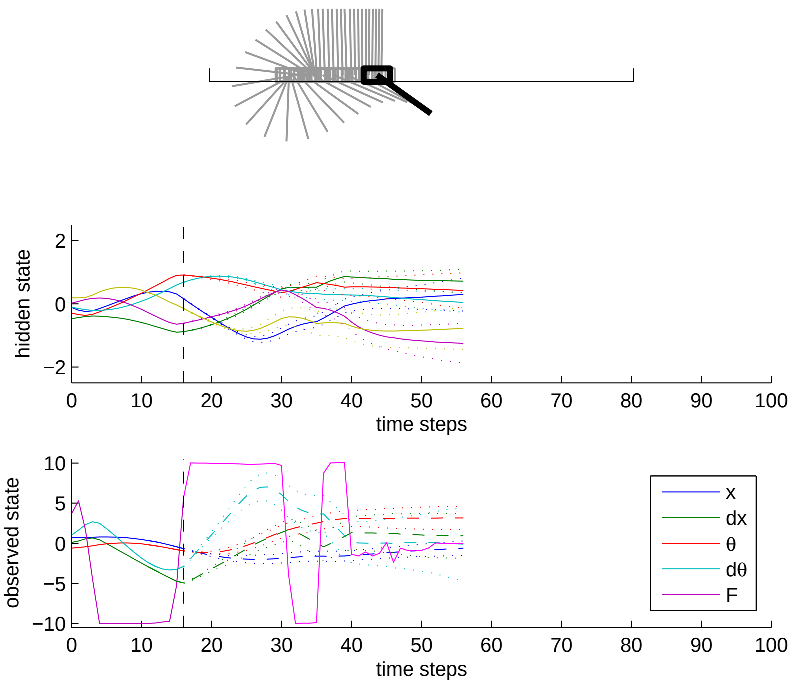


Figure 4.3: *Top:* The pole is successfully swung up by moving first to the left and then right. Predictions are plotted with grey. *Bottom:* The hidden states, observations, and the control signal in the same situation. The current time $t = 16$ is marked with a vertical dash line. The prediction horizon is 40 steps.

4.4 Bayes Blocks for nonlinear Bayesian networks

Nonlinear factor analysis and nonlinear state-space models can be seen as special cases of Bayesian networks where the linearity assumption is relaxed. Aside from these two, there are lots of possibilities to build the model structure that defines the dependencies between the parameters and the data. To be able to manage the variety, a modular software package using C++/Python called the Bayes Blocks (Valpola et al., 2003a) has been created. It is introduced in Publication I and in an earlier conference paper by Valpola et al. (2001).

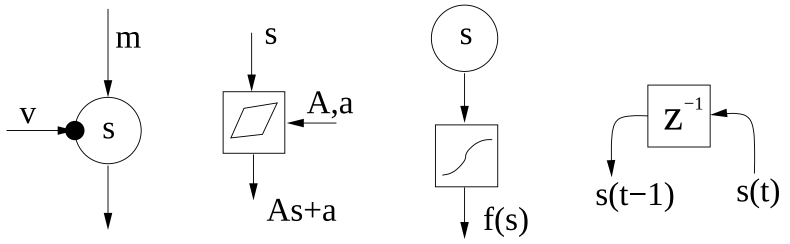


Figure 4.4: *First subfigure from the left:* The circle represents a Gaussian node corresponding to the latent variable s conditioned by mean m and variance $\exp(-v)$. *Second subfigure:* Addition and multiplication nodes are used to form an affine mapping from s to $As + a$. *Third subfigure:* A nonlinearity f is applied immediately after a Gaussian variable. *The rightmost subfigure:* Delay operator delays a time-dependent signal by one time unit.

The design principles for Bayes Blocks have been the following. Firstly, we use standardised building blocks that can be connected rather freely and can be learned with local learning rules, i.e. each block only needs to communicate with its neighbours. Secondly, the system should work with very large scale models. Computational complexity is linear with respect to the number of data samples and connections in the model.

The building blocks include Gaussian variables, summation, multiplication, and nonlinearity. The framework does not make much difference in parameters $\boldsymbol{\theta}$ and latent variables $\boldsymbol{S}$, the former are represented with scalars and the latter as vectors. Variational Bayesian learning provides a cost function which can be used for updating the variables as well as optimising the model structure. The derivation of the cost function, as well as learning and inference rules, is automatic which means that the user only needs to define the connections between the blocks.

The Gaussian node is a variable node and the basic element in building hierarchical models. Figure 4.4 (leftmost subfigure) shows the schematic diagram of the Gaussian node. Its output is the value of a Gaussian random variable s, which is conditioned by the inputs m and v. Denote generally by $\mathcal{N}\left(x; m_x, \sigma_x^2\right)$ the probability density function of a Gaussian random variable x having the mean m_x and variance σ_x^2. Then the conditional probability function of the variable s is $p(s \mid m, v) = \mathcal{N}\left(s; m, \exp(-v)\right)$. As a generative model, the Gaussian node takes its mean input m and adds to it Gaussian noise (or innovation) with variance $\exp(-v)$.

The addition and multiplication nodes are used for summing and multiplying variables. These standard mathematical operations are typically used to construct linear mappings between the variables. A nonlinear computation node can be used for constructing nonlinear mappings between the variable nodes. The delay operation can be used to model dynamics. Harva et al. (2005) implements several new blocks including mixture-of-Gaussians and rectified Gaussians. Harva and Kabán (2005) use the rectified Gaussian node to create a factor model with non-negativity constraints and Nolan et al. (2006) applies Bayes Blocks in an astronomical problem.

Nodes propagate certain expectations about their state to their neighbours in the network. For variable nodes in a network, update rules for the posterior approximation q that minimise the cost function $\mathcal{C}$ given that q of all the other variables stays constant, have been derived. The updates are very simple since the posterior approximation q is of a very simple form: It is a Gaussian with a diagonal covariance matrix.

Winn and Bishop (2005) introduce an algorithm called variational message passing with close similarities with Bayes Blocks. It does not allow for nonlinearities or variance modelling (see the following section), but on the other hand, it handles discrete variables more freely. It also allows for a posterior approximation factorised such that disjoint groups of variables are independent, but dependencies within the group are modelled. Variational message passing updates the posterior approximation of one factor at a time using VB learning. Note that the best properties of Bayes Blocks and variational message passing could be combined.

Similar models can be studied also with rather different posterior approximations. Spiegelhalter et al. (1995) introduce the BUGS software package that uses Gibbs sampling (see Section 2.5.4) for Bayesian inference. The package supports mixture models, nonlinearities, and non-stationary variance. A thorough experimental comparison to Bayes Blocks would be very valuable.

4.4.1 Variance modelling

In many models, variances are assumed to have constant values although this assumption is often unrealistic in practice. Joint modelling of means and variances is difficult in many learning approaches, because it can give rise to infinite probability densities. In Bayesian methods where sampling is employed, the difficulties with infinite probability densities are avoided, but these methods are not efficient enough for very large models. The Bayes Blocks allow to build hierarchical or dynamical models for the variance.

The Bayes Blocks framework was used by Valpola et al. (2004) to jointly model both variances and means in biomedical MEG data. The same approach can be used to translate any model for a mean to a model for a variance, so a large number of models in the literature could be explored as models for variance as well.

The left subfigure of Figure 4.5 shows how linear state-space model (see Section 3.1.6) is built using Bayes Blocks. It can be extended into a model for both means and variances as depicted graphically in the right subfigure of Figure 4.5. The variance sources $\mathbf{u}(t)$ characterise the innovation process of $\mathbf{s}(t)$, in effect telling how much the signal differs from the predicted one but not in which direction it is changing. Both regular sources $\mathbf{s}(t)$ and variance sources $\mathbf{u}(t)$ are modelled dynamically by using one-step recursive prediction model for them. The model equations are:

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}_x(t) \tag{4.14}$$

$$\mathbf{s}(t) = \mathbf{B}\mathbf{s}(t-1) + \mathbf{b} + \mathbf{n}_s(t) \tag{4.15}$$

$$n_{si}(t) = \mathcal{N}\left(n_{si}(t); 0, \exp\left[-u_i(t)\right]\right) \tag{4.16}$$

$$\mathbf{u}(t) = \mathbf{C}\mathbf{u}(t-1) + \mathbf{c} + \mathbf{n}_u(t), \tag{4.17}$$

where the variance of $n_{si}(t)$, the ith component of the noise vector $\mathbf{n}_s(t)$, is determined by the variance source $u_i(t)$.

4.4.2 Hierarchical nonlinear factor analysis

In hierarchical nonlinear factor analysis (HNFA) (Raiko, 2001; Valpola et al., 2003b), there are a number of layers of Gaussian variables, the bottom-most layer corresponding to the data. There is a linear mixture mapping from each layer to all the layers below it. The middle layer variables are immediately followed by a nonlinearity. The model structure for a three-layer network using Bayes Blocks is depicted in the left subfigure of Figure 4.6. Model equations are

$$\mathbf{h}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}_h(t) \tag{4.18}$$

$$\mathbf{x}(t) = \mathbf{B}\boldsymbol{\phi}[\mathbf{h}(t)] + \mathbf{C}\mathbf{s}(t) + \mathbf{b} + \mathbf{n}_x(t)\,, \tag{4.19}$$

where $\mathbf{n}_h(t)$ and $\mathbf{n}_x(t)$ are Gaussian noise terms and the nonlinearity $\phi(\xi) = \exp(-\xi^2)$ operates on each element of its argument vector separately. This activation function has the universal approximation property as well (see Stinchcombe and White, 1989, for proof). Note that the short-cut mapping $\mathbf{C}$ from sources to observations means that hidden nodes only need to model the deviations from linearity.

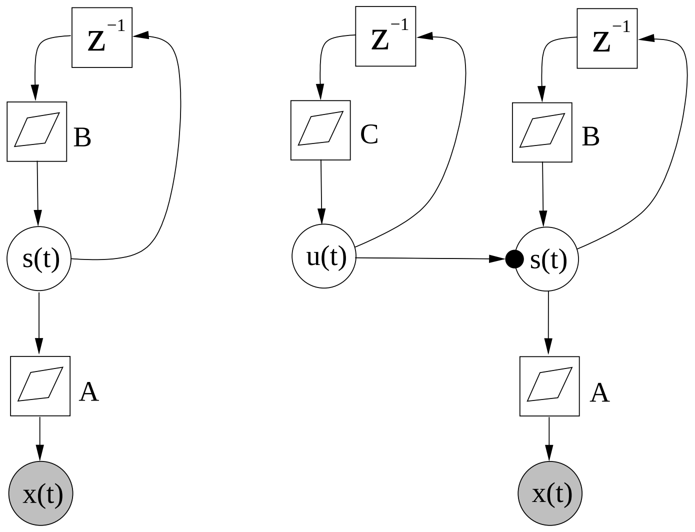


Figure 4.5: Model structures represented using the blocks in Figure 4.4. Observed variables are shaded. *Left:* A linear Gaussian state-space model. *Right:* A dynamic model for the variances of the sources which also have a recurrent dynamic model.

HNFA has latent variables $\mathbf{h}(t)$ in the middle layer, whereas in nonlinear FA, the middle layer is purely computational. This results in some differences. Firstly, the cost function $\mathcal{C}$ in HNFA is evaluated without resorting to approximation, since the required integrals can be solved analytically. Secondly, the computational complexity of HNFA is linear with respect to the number of sources, whereas the computational complexity of nonlinear FA is quadratic. HNFA is thus applicable to larger problems, and it is feasible to use even more layers than three. Also, the efficient pruning facilities of Bayes Blocks allow determining whether the nonlinearity is really needed and pruning it out when the mixing is linear, as demonstrated by Honkela et al. (2005).

The good properties of HNFA come with a cost. The simplifying assumption of diagonal covariance of the posterior approximation, made both in nonlinear FA and HNFA, is much stronger in HNFA because it applies also in the middle layer variables $\mathbf{h}(t)$. Publication II compares the two methods in reconstructing missing values in speech spectrograms. As seen in the right subfigure of Figure 4.6, HNFA is able to reconstruct the spectrogram reasonably well, but quantitative comparison reveals that the models learned in HNFA are more linear (and thus in some cases

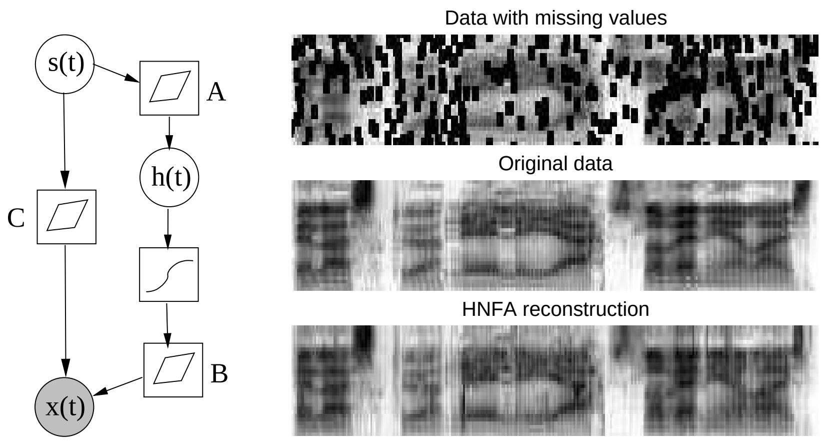


Figure 4.6: *Left:* The model structure for hierarchical nonlinear factor analysis (HNFA). *Right:* Some speech data with and without missing values (Setting 1) and the reconstruction given by HNFA.

worse) compared to the ones learned in nonlinear FA.

4.4.3 Relational models

So far, the models have been divided into two categories: static and dynamic. In static modelling, each observation or data sample is independent of the others. In dynamic models, the relations between consecutive observations are modelled. The generalisation of both is that the relations are described in the data itself, that is, each observation might have a different model structure. The following two chapters concentrate on relational models. One of the models, nonlinear relational Markov network (see Section 6.3), is implemented using Bayes Blocks.

Chapter 5

Inductive logic programming

Often the structure that relates objects or variables in machine learning tasks is assumed to be constant, for example, the data comes in samples of fixed size as in all of the models presented in Section 3.1. Sometimes the samples are structured, like molecules, or related to each other in an individual manner, like web pages. First-order logic, or equivalently, relational modelling is needed to represent such structured data. Inductive logic programming aims at learning logic programmes from data by combining machine learning and first-order logic, but let us first discuss logic programming in general.

5.1 Logic programming

The main idea of logic programming is that deduction can be viewed as a form of computation (Sterling and Shapiro, 1994). The declarative statement

$$H \Leftarrow B_1 \wedge B_2 \wedge B_3 \tag{5.1}$$

can be interpreted procedurally as "to solve H, solve B_1 and B_2 and B_3", or shortly $H \leftarrow B_1, B_2, B_3$. A logic programme is thus a set of logical axioms and it is run by querying for a proof of some goal statement. The closed world assumption (Reiter, 1978) is used, that is, everything that cannot be proven to be true, is assumed to be false.

Normally the statements in a logic programme are restricted to be of the form $H \leftarrow B_1, \ldots, B_n$, that is, they are so-called Horn clauses. This ensures that the proof for the goal statement has a simple tree structure. Atom H is called the head of the clause and the atoms $B_1, \dots, B_n$ form the body of the clause. Statements with $n = 0$ are called facts because the proof of the head does not require solving any more statements.

Logic programming uses first-order logic. This means that an atom can be structured as a predicate followed by a number of arguments in brackets. Some of the arguments can be variables.[1] An example of a rule (or clause) that uses first-order logic is $\text{son}(X, Y) \leftarrow \text{parent}(Y, X), \text{male}(X)$, that is, X is the son of Y if Y is the parent of X and X is male. X and Y are logical variables that can represent any object (people in this case). Variables are written in upper case to avoid confusion. Atoms that do not contain any variables are called ground. Completeness theorem by Gödel (1929) states that in first-order predicate calculus every logically valid formula is provable. Second-order logic allows variables as predicates, but completeness does not hold anymore.

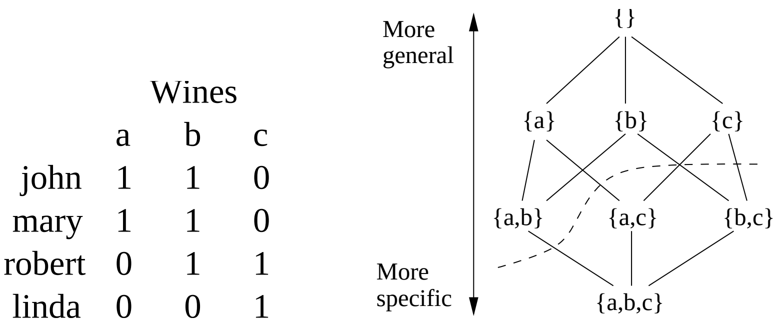
A logic programme that contains the rule $\text{son}(X, Y) \leftarrow \text{parent}(Y, X), \text{male}(X)$ and some ground facts such as $\text{parent}(mary, robert)$ and $\text{male}(robert)$, can answer a query $\text{son}(X, mary)$. The solver finds the rule and tries to solve the body of the rule, that is $\text{parent}(mary, X)$ and $\text{male}(X)$. The only solution is $X = robert$ which is returned as the output of the programme.

Prolog is the best-known logic programming language. Sterling and Shapiro (1994) wrote a good book on logic programming and Prolog in specific. Logic programming differs somewhat from traditional procedural programming. The clearest difference is that the values of variables are fixed once set. Where procedural programmes tend to have for-loops, logic programmes use recursion instead. The strong areas of logic programming include handling structured data, symbolic manipulation, and self-changing programmes. In inductive logic programming, rules in logic programmes are learned from examples.

5.2 Inductive logic programming

Inductive logic programming (ILP) provides tools for relational data mining, that is, mining from data stored in multiple tables. It works with the powerful language of logic programmes, both as prior domain knowledge and as describing the discovered patterns. A good introduction to the theory, implementation, and

[1]Atoms may in general be nested. For example $\text{knows}(X, \text{mother}(X, Y)) \leftarrow \text{mother}(X, Y)$ means that every person X knows that she is the mother of Y if that really is the case. Nested atoms are out of the scope of this thesis.



	Wines		
	a	b	c
john	1	1	0
mary	1	1	0
robert	0	1	1
linda	0	0	1

Figure 5.1: *Left:* Wine tasting data where 1 means that the person liked the wine. Right: The hypothesis space includes all combinations of items (wines) ordered in a trellis where the edges represent minimal generalisation (upwards) or minimal specialisations (downwards). The dashed curve represents the border between frequent and infrequent itemsets, assuming 30% frequency threshold.

applications of ILP is written by Muggleton and De Raedt (1994). Another introduction to ILP that also relates logic programming terminology to database terminology, is given by Dzeroski and Lavrac (2001). Books that address ILP have been written by De Raedt (2005, 1996); Lloyd (2003), and Furukawa et al. (1999).

The basic data mining task of ILP is as follows: Given positive (and possibly negative) examples, a concept description language, and possibly background knowledge, find a set of association rules that covers most of the positive examples but only few of the negative examples.

5.2.1 Example on wine tasting

Let us first study a simple example of propositional data mining in the domain of wine tasting. The task is to recommend wines based on the list of other wines that a person likes. Let us assume that we have a large database with information of whether or not some people like a particular wine. This can be represented as a table with wines as columns and people as rows. Each cell contains a 1 if the person likes the wine and 0 if not. Such a table is shown in Figure 5.1.

First, we will find interesting sets of wines. We measure how often all the wines of the set are liked by the same people. A frequent itemset is a set of columns for which the number of rows that has only 1s in the corresponding cells is greater than some threshold. Let us say that a group of wines is a frequent itemset if at least 30% of the people like all of them. In this case, the frequent itemsets are $\{\}, \{a\}, \{b\}, \{c\}$, and $\{a, b\}$.

The hypothesis space for different itemsets (or patterns) is depicted in Figure 5.1. The different hypotheses have a partial order for generality and specificity. If an itemset is frequent, all itemsets that are generalisations of it, are also. If an itemset is not frequent, all itemsets that are specialisations of it, are infrequent as well. This property is essential for pruning the hypothesis space during search. For instance, if we know that $\{a, c\}$ is not frequent, we also know that $\{a, b, c\}$ is not frequent without testing. The size of the hypothesis space is exponential with respect to the number of items, so pruning is essential to achieve a reasonable computational complexity.

An association rule tells that if a person likes a certain set of wines, he or she will like some other wine. A frequent itemset can be transformed into an association rule by choosing one of the wines to be the one to be predicted based on the others. Now we can test whether the rule applies to all the people in the database, or is statistically significant. The found rules are the logic programme that were inferred from the data inductively. In the example, we can find the rule $b \leftarrow a$ that applies to all cases, that is, everyone who likes wine a also likes wine b.

5.2.2 From propositional to relational learning

The wine tasting example is simple: The data consist of a single table and each row had only one index (the name of the person). This case is called attribute-value learning or propositional learning. The case with relational data in multiple tables and with multiple indices is more complex. In the wine example, first we need to reformulate $b \leftarrow a$ as $\text{likes}(X, b) \leftarrow \text{likes}(X, a)$, that is, every person X who likes a, also likes b. Then, we could also have knowledge of marriages between people, that is, $\text{husband}(X, Y)$ is true iff X is the husband of Y. Now the hypotheses include clauses such as $\text{likes}(X, Y) \leftarrow \text{husband}(X, Z), \text{likes}(Z, Y)$, that is, every husband X likes all the wines Y that his wife Z likes. We could also know the grape and origin of each wine and make a hypothesis that anyone who likes a wine that is made of Pinot Noir likes all wines from the same origin. The hypothesis space becomes more complex, but the trellis defined by the generality relationships is still present as is.

For traversing the hypothesis space, refinement operators are used. One is the most general specialisation, mgs($\cdot$), that corresponds to an edge downwards in the lattice of Figure 5.1. Let us define that $D > C$ means that D is more specific than C. Hypothesis $D \in \text{mgs}(C)$ iff $D > C$ and there is no hypothesis E such that $D > E > C$. For example, hypothesis $\{a, b\}$ is more specific than hypothesis $\{a\}$ since everyone who likes both wines a and b trivially like wine a. The hypothesis $\{a, b\}$ is also a most general specialisation of $\{a\}$ since there is no other hypothesis that would fit between these two. Note that a hypothesis may have more than one most general specialisation. The least general generalisation, lgg($\cdot$), is the inverse of mgs($\cdot$). It corresponds to an edge upwards in the hypothesis lattice. $D \in \text{lgg}(C)$ iff $D < C$ and there is no E such that $D < E < C$.

One can generate all (possibly infinite) hypotheses in the hypothesis space if one applies the most general specialisation operation to the null hypothesis repeatedly. Two of such systems include FOIL by Quinlan (1990) and PROGOL by Muggleton (1995). Some ILP systems, such as GOLEM by Muggleton and Feng (1992) and Aleph by Srinivasan (2005), start from the most specific hypotheses and work their way upwards using the least general generalisation, and some ILP systems use both types of refinement operators. There are dozens of ILP systems listed on the web page[2] of Network of Excellence in Inductive Logic Programming ILPnet2.

5.2.3 Applications

ILP is often applied to data mining tasks. The goal is not always just concept learning, as presented above. It is also possible to perform classification, distance based learning, clustering, descriptive learning, kernel based learning, reinforcement learning, and so on. Here are two example applications.

Toxicological databases list molecules and their effects to living organisms. It is possible to use ILP to predict this activity based on the structure of the molecule. The structure can be represented as relational tables containing atoms and bonds. Itemsets are the frequent substructures in the molecules that should be useful in classification. Figure 5.2 shows an example. Helma et al. (2000) test several different ILP systems on predictive toxicology. Graph based molecular data mining (see overview by Fischer and Meinl, 2004) is an active research topic with lots of unsolved questions.

Intrusion detection systems are used to monitor computer systems for signs of security violations. Normally the alerts are presented to the human analyst, but Pietraszek and Tanner (2005) present an application of ILP for automatic classifi-

[2]`http://www.cs.bris.ac.uk/~ILPnet2/`

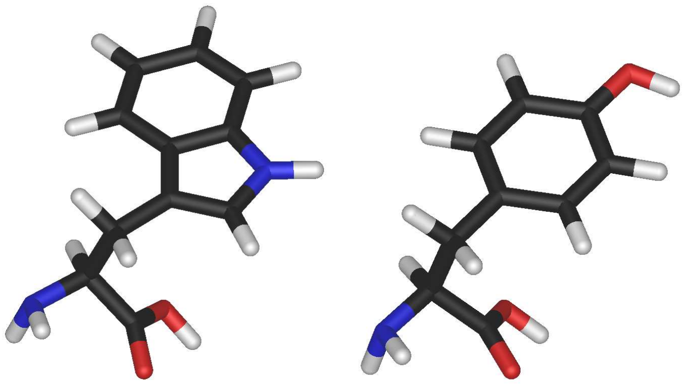


Figure 5.2: Two molecules share a substructure in the bottom left arms. Substructures are useful in classifying molecules.

cation of alerts to decrease the number of false alerts. The data contain the time of day and week, source and destination ports and IP addresses of the connection, as well as the amount of traffic for each alert. There is also some background knowledge such as the network topology. Other alerts related to the current one are essential in some cases such as password guessing and port scanning. The induced rules were comprehensible and could decrease the number of false alarms considerably.

Chapter 6

Statistical relational learning

Chapters 3 and 4 study machine learning from data containing discrete and continuous values. Statistical relational learning or probabilistic logic learning adds another element: *References* are used to describe relationships between objects. For example, the contents of a web page can be described by a number of attributes, but the links between pages are important as well. Taskar et al. (2002) show that using relational information in classification of web pages makes the task much easier.

First-order logic is one way of handling references (or relations). Statistical relational learning can also be seen as an extension of inductive logic programming (ILP) described in Chapter 5. The motivation of upgrading ILP to incorporate probabilities is that the data often contain noise or errors, which calls for a probabilistic approach.

There is a large body of work concerning statistical relational learning in many different frameworks. See (De Raedt and Kersting, 2003) for an overview and references. Two of these frameworks will be briefly reviewed in the following.

The formalism of probabilistic relational models (PRMs) by Koller (1999); Getoor et al. (2001) provides an elegant graphical representation of objects, attributes, and references (see Figure 6.1). Perhaps its close analogy with relational databases and object oriented programming has made the formalism quite popular. A PRM consists of two parts: the relational schema for the domain and the probabilistic component. Given a database, the relational schema defines a structure for a Bayesian network over the attributes in the data. The probabilistic component

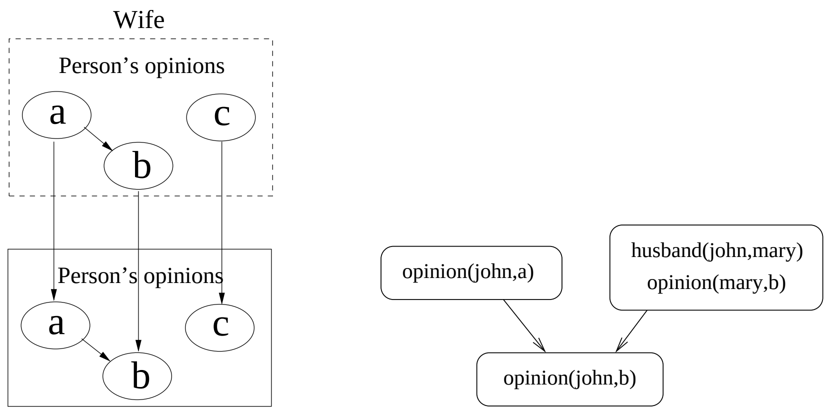


Figure 6.1: Left: The graphical representation of a probabilistic relational model. Right: The parents of opinion$(john, b)$ in a Bayesian network created by a Bayesian logic programme when applying rules opinion$(X, b) \leftarrow$ opinion(X, a) and opinion$(X, Y) \leftarrow$ husband(X, Z), opinion(Z, Y). The full network is not shown.

describes dependencies among attributes, both within the same object and between attributes of related objects. The conceptual simplicity comes with a cost in generality. For instance probabilistic dependencies between links are harder to represent and require special treatment (Getoor et al., 2002). PRMs have been applied for instance to gene expression data by Segal et al. (2001).

Kersting and De Raedt (2001, 2006) introduced the framework of Bayesian logic programmes (BLPs). BLPs generalise Bayesian networks, logic programming, and probabilistic relational models. Each atom has a random variable associated to it. For each clause, there is the conditional distribution of the head given the body. The proofs of all statements form a (possibly infinite) Bayesian network where atoms are nodes and the head atom of each clause is a child node of the nodes in the body of the clause. Again, given the logical part of the data, a BLP forms a Bayesian network over the attributes of the data. Bayesian networks must be acyclic so the same applies to both PRMs and BLPs.

Let us continue the wine tasting example in Chapter 5 and discuss it from a BLP point of view. Instead of likes(X, Y) atoms we use opinion(X, Y) and associate a variable (attribute) to the atom that actually tells what the opinion is. Then the rule opinion$(X, b) \leftarrow$ opinion(X, a) means that the opinion on wine

b depends on the same person's opinion on wine a. The rule opinion$(X, Y) \leftarrow$ husband(X, Z), opinion(Z, Y) tells that the opinion about a wine depends on the wife's opinion on the same wine. The actual probabilistic dependencies are placed in conditional probability distributions associated to each rule. Note that whereas a rule in ILP can only make the atom likes(X, Y) true but never false, a rule in BLP can change the opinion to good or bad.

6.1 Combination rules

There is one non-trivial point in forming a Bayesian network from a PRM or a BLP. It is when there are one-to-many or many-to-many relationships or equivalently multiple proofs for a single atom. The child node in the Bayesian network would get many sets of parents where each set defines a conditional probability distribution for the child. The number of parents varies from sample to sample. This is solved by combination rules (or combining rules or aggregate dependencies or aggregate functions) which combine many probability distributions into one.

Figure 6.1 shows a situation where two rules apply to opinion$(john, b)$. Each rule gives a conditional probability for John's opinion about wine b and they must be combined using a combination rule. Note that the number of rules that apply, varies from sample to sample.

The most typical combination rule is the *noisy-or* (see Pearl, 1988) for binary variables. The probability of the binary variable x being false given its binary parents $\mathbf{y} = (y_1, \ldots, y_n)$ is $P(x = 0 \mid \mathbf{y}) = \prod_{i|y_i=1} q_i$, that is, x is false iff all its possible causes y_i are independently inhibited by noise with probability q_i each. For example, each disease y_i has a probability $1 - q_i$ to cause fever x and the patient gets the fever if any one of the diseases cause it.

Noisy-or is asymmetric with respect to the binary variables it deals with. If zeros and ones are mutually exchanged, the rule becomes noisy-and. Publication VI studies two combination rules that are symmetric and can be applied to discrete and continuous values as well. The first is Naïve Bayes (or maximum entropy) combination rule which corresponds to having a Markov network where the different sets of parents are not connected to each other. The second is product of experts, where the probability density is a function of the product of probability densities proposed by the different experts (or sets of parents). Other combination rules include sigmoid (Neal, 1992), noisy maximum and minimum (Diez, 1993), mixture of experts, and any aggregate functions such as sum, average, median, mode, and count (Getoor, 2001; Kersting and De Raedt, 2006).

6.2 Logical hidden Markov models

Logical hidden Markov models (LOHMMs) introduced in Publication VII, deal with sequences of structured symbols in the form of logical atoms, rather than flat characters. They can be seen as a special case of statistical relational learning or as an extended version of hidden Markov models (HMMs, see Section 3.1.5). LOHMMs try to retain as much of the expressiveness of first-order logic as possible while still having as efficient algorithms for learning as HMMs. States and observations are logical atoms and transitions can involve variable bindings. LOHMMs have been implemented in Prolog.

Many real world sequences such as protein secondary structures or UNIX shell logs exhibit a rich internal structure. Traditional probabilistic models of sequences, however, consider sequences of unstructured symbols only. For instance, consider a sequence of UNIX commands, which may have parameters such as “emacs lohmms.tex, ls, latex lohmms.tex”. Commands are essentially structured. Applying HMMs requires either 1) ignoring the structure of the commands (i.e., the parameters), or 2) taking all possible parameters explicitly into account. The former approach results in a serious information loss and the latter leads to a combinatorial explosion in the number of symbols and parameters of the HMM and as a consequence inhibits generalisation. Using logical atoms, the above UNIX command sequence can be represented as “emacs(lohmms.tex), ls, latex(lohmms.tex)”. There are two important motivations for using logical atoms at the symbol level. Firstly, variables in the atoms allow one to make abstraction of specific symbols. For example, the logical atom emacs(X) represents all files X edited using emacs. Secondly, unification allows one to share information among states. For example, the sequence emacs(X), latex(X) denotes that the same file is used as an argument for both emacs and latex.

Let us return to the coin example in Section 3.1.5 to help define a LOHMM as a generative model. Figure 6.2 shows the HMM and the corresponding LOHMM. Transitions are divided into two steps (compared to just one in HMMs). To generate data, first, a rule (or an abstract transition) is selected according to the (abstract) transition probabilities to find out the resulting abstract state (an atom such as coin(X)). Only the rules with most specific bodies are applied, for instance in the state coin(2), the rules for coin(X) are not used. Second, the remaining variables in the atom are instantiated using so called selection probabilities. The scope of variables is restricted to single transitions, that is, abstract states coin(X) and coin(Y) are equivalent after variable bindings in that transition are taken into account. Note that also the observations (h and t) can be structured atoms in general.

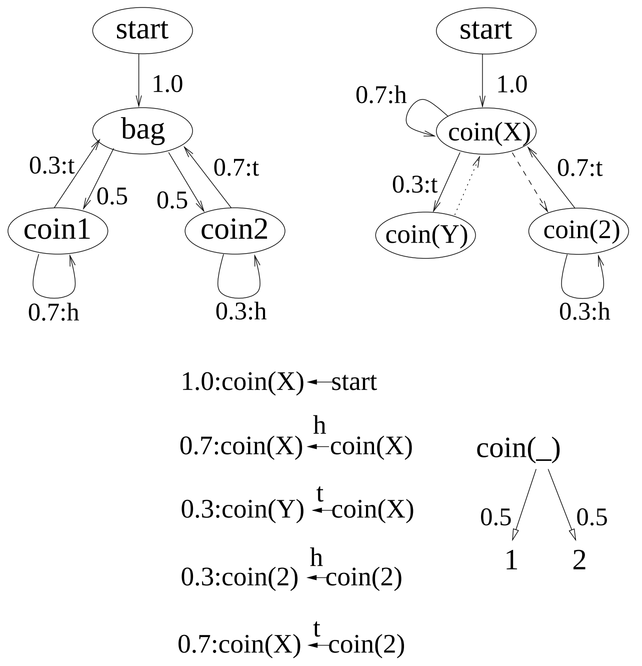


Figure 6.2: *Left:* A graphical representation of a hidden Markov model repeated from Figure 3.3. *Right:*The corresponding logical hidden Markov model. *Bottom:* The logical hidden Markov model written as a logic programme. Solid arrows are abstract transitions, dashed arrows denote special cases, and dotted edges are needed because of the scope of variables. White-headed arrows in the bottom right show the selection probabilities.

The rules of a LOHMM form a logic programme where the only fact is the start state. The proof network for the observation sequence forms a structure of the corresponding graphical model. Figure 6.3 depicts a graph formed by unfolding a tiny LOHMM in the UNIX command sequence modelling. Using the graph it is possible to see how to generalise inference algorithms of HMMs. As for HMMs, three inference problems are of interest. Let $\mathcal{H}$ be a LOHMM and let $\mathsf{X} = \mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_T$, $T > 0$, be a finite sequence of ground observations:

Evaluation: Determine the probability $P(\mathsf{X} \mid \mathcal{H})$ that sequence X was generated by the model $\mathcal{H}$.

Most likely state sequence: Determine the hidden state sequence that has most likely produced the observation sequence X.

Parameter estimation: Given a set $\{\mathsf{X}_1, \ldots, \mathsf{X}_k\}$ of observation sequences, determine the most likely parameters for the abstract transitions and the selection distributions of $\mathcal{H}$.

Publication VII addresses each of these problems in turn by extending the existing solutions for HMMs. Belief propagation (see Section 3.1.1), also known as the forward-backward algorithm in the context of HMMs, is used to compute the probability of a particular abstract and ground transition at a particular time, given parameters. Belief propagation, as well as evaluation and finding the most likely hidden state sequence have the computational complexity of $\mathcal{O}(Ts^2)$ where T is the data size and s is the number of possible states.

Probabilities found by belief propagation can be further used for parameter updating by summing over time to get expected counts for both abstract transitions and selections. Raiko et al. (2002) explain how this ML parameter estimation is transformed into a more Bayesian solution. The computational complexity is $\mathcal{O}(I(Ts^2+d))$ where I is the number of iterations and d is the number of parameters in the model. Experiments (see Section 6.2.3) demonstrate that LOHMMs possess several advantages over traditional HMMs for applications involving structured sequences.

6.2.1 Reachable states

An important point for computational efficiency in the parameter learning algorithm is the pruning of unreachable states. Before any probabilistic parameters are even considered, the algorithm finds all the reachable hidden states at each time step, given the whole observation sequence. The algorithm works as follows.

The only reachable state at time 0 is the *start* state. Then for each time t from 0 to $T-1$, all the transitions from S_t that agree with the current observation $\mathtt{x}_t$ are used to produce the set of reachable states S_{t+1}. At this stage, the states are reachable given the observation sequence so far. Finally for each time t back from $T-1$ to 0, those states in S_t whose transitions lead outside S_{t+1}, are removed. This takes into account the whole observation sequence.

Note that this procedure resembles the forward-backward algorithm for HMMs. As a by-product, it can be used to check whether an observation sequence could

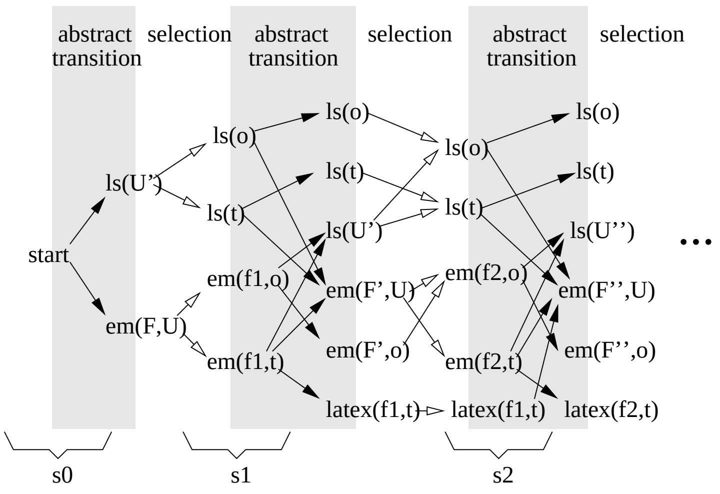


Figure 6.3: A logical hidden Markov model is unfolded in time to form a trellis. Transitions are factorised into two steps, abstract transitions (rules) and selection (variable instantiation). The example represents user modelling where the states include commands ls, emacs, and latex, and the user type (t or o) is included as part of the hidden state. Filenames f1 and f2 are the other arguments of emacs and latex. See Publication VII for details.

have been generated with the LOHMM. If not, the sets of reachable states are empty.

6.2.2 Structural learning

The increase in expressiveness of LOHMMs over traditional HMMs comes at the expense of a more complex model selection problem. Indeed, different abstraction levels have to be explored. Publication IX proposes a novel method for selecting logical hidden Markov models from data. The proposed method adapts structural expectation maximisation (EM) by Friedman (1997). It combines a generalised expectation maximisation algorithm, which optimises parameters, with structure search for model selection using inductive-logic-programming (ILP) refinement operators. Structural learning of traditional HMMs has not been very popular, only recently Won et al. (2006) applied genetic algorithms for that.

Given a set $\{\mathsf{X}_1, \ldots, \mathsf{X}_k\}$ of observation sequences, a (possibly infinite) set of LOHMMs structures, and a scoring function, **find** the model structure that maximises the score.

Selecting a structure of a LOHMM is a significant problem for many reasons. Firstly, extracting structures from experts can be a laborious and expensive process. Secondly, HMMs are commonly learned by estimating the maximum likelihood parameters of a fixed, fully connected model. Such an approach is not feasible for LOHMMs as different abstraction levels have to be explored. Finally, the parameter estimation of a LOHMM is a costly nonlinear optimisation problem, so the naïve search is infeasible.

The idea behind structural EM is to first infer the distribution over the hidden states and collect sufficient statistics about it. In the case of LOHMMs the sufficient statistics are the expected counts of how many times a ground transition is used. Then different model structures are evaluated based on those statistics. Evaluating new structures is thus made independent of the number and length of the data cases — a feature which is important for scaling up.

6.2.3 Applications

LOHMMs have been applied to several different problems. Publication VIII addresses the application to protein-fold recognition. The number of determined protein structures is growing rapidly and there are different classification schemes for them. There is a need for computer methods that can automatically extract structural signatures for classes of proteins. The secondary structure of a protein is represented as a sequence of structured symbols, so applying LOHMMs is very natural. The results on the database and classification scheme SCOP (Structural Classification Of Proteins from Murzin et al. (1995)) indicate that LOHMMs possess several advantages over other approaches.

Another application of LOHMMs in the biological domain is the mRNA signal structure detection presented in Publication VII. mRNA molecules fold to form a secondary structure which can be described with concepts such as stacking regions, hairpin loops, and interior loops. The secondary structure of an mRNA forms a tree which makes it more challenging than that of a protein. A LOHMM was used to parse a tree in in-order (the node itself between its children) while the tree structure is essentially stored in the arguments of the hidden state. Classification accuracy was higher than with the comparison method by Horváth et al. (2001).

UNIX command sequences have been studied with LOHMMs in Publication IX.

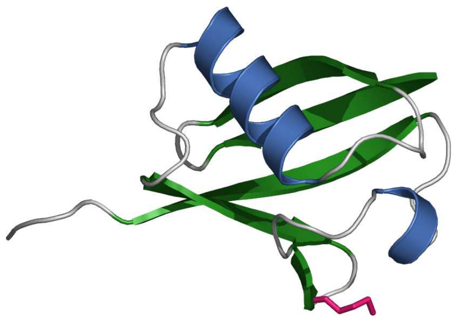


Figure 6.4: A small protein fold represented emphasising the secondary structure with helices (blue) and strands (green).

Tasks that have been considered for UNIX command sequences include the prediction of the next command in the sequence by Davison and Hirsh (1998), the classification of a command sequence in a user category by Korvemaker and Greiner (2000); Jacobs and Blockeel (2001), and anomaly detection by Lane (1999). LOHMMs could be applied to all of these tasks and Publication IX reports experiments in the classification task with results comparable to other methods.

Landwehr et al. (2006) use a custom implementation of LOHMMs for haplotype reconstruction from genotype data. The proposed method offers a competitive trade-off between accuracy and computational complexity compared to other state-of-the-art systems developed for the task.

6.3 Nonlinear relational Markov networks

Bayesian networks assume acyclicity of the network structure. The directed edges in the graph can be interpreted as causal dependencies and nothing can cause itself. The same assumption is inherited by BLPs and PRMs.[1] In some cases it is difficult or irrelevant to try to model the direction of the dependency. Say, whether the husband adopts opinions from his wife, or vice versa, or whether people with certain combinations of opinions are more likely to marry. Using directed edges for describing friendship would definitely lead into cycles with a group of friends. Markov networks (see Section 3.1.2) model dependencies with undirected edges so

[1]LOHMMs are acyclic by definition since all directed edges point from past to future.

that it only tells that there is a dependency but not what causes what.

Relational Markov networks (RMN) by Taskar et al. (2002) are to Markov networks what BLPs are to Bayes networks. A RMN is specified by a set of clique templates (the logical part) and a potential for each clique template (the probabilistic part). For instance, the probabilistic part of the template (opinion(X, Y), husband(X, Z), opinion(Z, Y)) could describe how the opinions of the husband X and wife Z about the wine Y are related. Given a relational database, the RMN produces an unrolled Markov network over all the attributes in the data. The cliques instantiated by a certain template share the same clique potential. Note that an RMN does not require explicit combination rules.

The general inference task in RMNs is to compute the posterior distribution over all the attributes. The network induced by data can be very large and densely connected, so exact inference is often intractable. The loopy belief propagation algorithm by Murphy et al. (1999) (see Section 3.1.1) is used as an approximation. The learning task, or the estimation of the clique potentials, requires alternating between updating the parameters of the potentials and running the inference algorithm on the unrolled Markov network.

Nonlinear relational Markov networks (NRMN), introduced in Publication VI, combine the ideas of relational Markov networks by Taskar et al. (2002) and nonlinear Markov networks (NMN) by Hofmann and Tresp (1998) (see Section 3.1.2). The combination is not very straightforward because the models are quite different: RMN specifies potentials over cliques of the network whereas NMN specifies a distribution of each variable given its neighbours in the network. In NRMN, the first approach is chosen.

Recall Figure 3.1 that shows a Markov network and its join tree. A node in the join tree corresponds to a clique in a Markov network. NRMN defines a probability distribution over the attributes in each clique template. The distributions are provided by HNFA described in Section 4.4.2. The maximum entropy combination rule requires marginalisation of probability distributions. In nonlinear models, this cannot usually be done exactly and therefore another combination rule was selected. In the product-of-experts (PoE) combination rule, the probability density is the average of the incoming probability densities on the logarithmic scale. A characteristic of PoE is that implicit weighting happens in some sense automatically. When one of the experts gives a distribution with high entropy (little information) and another one with low entropy (much information), the combination is close to the latter one.

NRMNs extend graphical models in both nonlinear and relational directions at the same time. Convergence is guaranteed regardless of loops, unlike in the loopy BP algorithm. There is a lot of room for improvement, though. The current version of NRMN includes many simplifying assumptions, such as diagonality of the posterior covariance matrix in HNFA, and separate learning of experts. Experiments with the game of Go (see Figure 6.5) give promise for NRMNs.

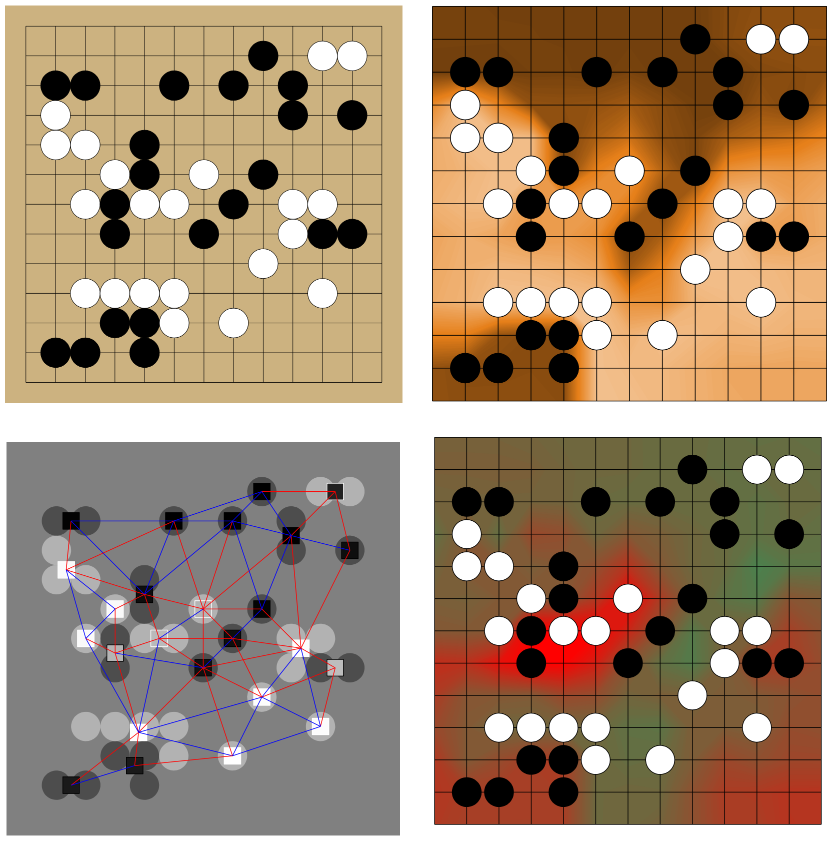


Figure 6.5: *Top left:* The board of a Go game in progress. Two players alternately place stones on empty points trying to surround area and opponent stones. *Top right:* The expected owner of each point is visualised with the shade of grey. For instance, the two white stones in the upper right corner are very likely to be captured. *Bottom left:* The strings of stones with their expected owner as the colour of the square. Pairs of related strings are connected with a blue line if the blocks have same colours and with a red line when the blocks have opposing colours. The lines also represent the structure of the implied Markov network. *Bottom right:* The covariance between owning a point and scoring high can be used to determine which parts of the board are important (red). The study of this kind of data is left as future work.

Chapter 7

Discussion

Extending graphical models to different directions provides a framework where an ever increasing number of machine learning methods fit. Some people oppose general solutions in principle, as problem-specific solutions are often more efficient in practice. A general framework, on the other hand, gives many benefits. Let us think of a speech recognition system consisting of three modules: the first converts an audio stream to phonemes, the second stacks phonemes into words, and the third stacks words into sentences. The communication of uncertainty between modules becomes an important point. If all the modules are built as graphical models, this interaction is straightforward and well founded. Secondly, the same methods can be used to analyse DNA sequences as well as phoneme sequences. A general framework, such as the one introduced in Publication I, allows reuse of ideas and software between sometimes surprisingly different applications.

Sometimes it is also reasonable to step back from generality and study useful special cases. For instance in statistical relational learning, most attention has been devoted to highly expressive formalisms. Logical hidden Markov models, introduced in Publication VII, can be seen as an attempt towards downgrading such highly expressive frameworks. They retain most of the essential logical features but are easier to understand, adapt, and learn. For the same reasons, simple statistical techniques (such as logistic regression or naïve Bayes) have been combined with ILP refinement operators for traversing the search space (see e.g. Popescul et al., 2003; Landwehr et al., 2005). In nonlinear modelling, special cases such as nonlinear state-space models, allow for specialised algorithms for initialisation, visualisation, and inference. Publication V presented an algorithm to speed up inference in nonlinear state-space models.

Computational complexity plays an important part in the methods presented in this work. Whereas the time complexity of some methods scale exponentially w.r.t. the size of the problem, the methods studied here scale linearly or quadratically. This allows for tackling relatively large problems. For instance, the dimensionality was hundreds in Publication I and the number of possible states was again hundreds in Publication VIII. In small problems, where even exponential computational complexity is not prohibitive, the methods studied here do not probably give the most accurate results.

The learning and inference algorithms presented in this work concentrate on a single solution candidate with its neighbourhood. This approach is good for its computational efficiency but it is prone to bad local optima. In many problems such as tracking (Särkkä et al., 2006), it is very important to explore many different solutions. It is possible to keep track of several solution candidates at the same time and during adaptation, to move bad candidates to the vicinity of a better ones. This same idea is used in beam search, particle filters (Doucet et al., 2001), and genetic algorithms.

Studying machine learning can also help in understanding how the human mind works. In the brain, most of the interaction is local, in the sense that the brain cells directly affect only those cells with which they are in contact. Some machine learning methods like the belief propagation and the Bayes Blocks framework, share this notion, while others, such as line search in an optimisation of a global cost function, do not. Some people would thus prefer the former. It is of course true that machine learning does not have to work by the same principles as biological brains, but local algorithms have the benefit of being parallelisable.

7.1 Future work

Perhaps the most important suggestion for future work is to bring lessons learned from the special cases of nonlinear state-space models and logical hidden Markov models back to the more general frameworks. Both have good algorithms for learning and inference that could be generalised.[1] The method for nonlinear state-space models includes properties such as posterior dependencies and control, that have not been implemented in the otherwise more flexible Bayes Blocks framework.

The visualisation of the learning process could help understanding the methods better, as well as help to find better initialisations, model structures, or means to avoid local minima. This is especially important for new users who do not know

[1]The algorithmic improvements in nonlinear state-space models are ongoing.

the methods well. Also general usability in most methods needs improvement so that potential users become users at all.

The number of node types in the Bayes Blocks framework could be increased. Feasible blocks not presented here include discrete variables, the error function nonlinearity (see Frey and Hinton, 1999), the absolute value, the maximum function (adapt Harva and Kabán, 2005), and MLP networks. The posterior dependencies of Gaussian variables could be handled relatively easily if the clique size of the join tree (see Figure 3.1) stays reasonable. If the clique size is too large, it is possible to use dummy random variables that have posterior correlations with other variables but no other role in modelling. The framework could also allow parallel processing. The assumption that vectorised nodes have the same length and they all have the same parents restrict their use in relational models, whereas scalar nodes have a lot of overhead and are thus inefficient when used to emulate more flexible vector nodes.

In some applications, the components of the data have coordinates, like the pixels of an image in computer vision. A latent variable could refer to the coordinates, as is done for instance by Winn and Joijic (2005). In another example, changing the pitch of a voice moves it vertically in the spectrogram. It would be quite reasonable to model the place of an object or a pitch of a voice with latent variables, but MLP networks would not be well suited to model the mapping to observations. It would be important to be able to model these rather different kinds of nonlinear mappings compared to the ones used in this thesis.

All the learning methods in this thesis aim at unsupervised learning where all the data is modelled with equal interest. When it is known beforehand how the model is going to be used, one could concentrate the learning efforts to the task at hand. This related to attention in cognitive modelling, and discriminative learning (see Taskar et al., 2002, for an example) in machine learning. Even better, Lasserre et al. (2006) introduce a principled hybrid of generative and discriminative models.

More applications are needed to show the full potential of the studied methods. Nonlinear state-space models could easily be used as feature extraction in speech recognition. An interesting application for relational models would be to study library data including title, contents, lending history, classification, and keywords for the material. The found model could be then applied to find structure in web pages. The application to the game of Go could also be continued. An experimental comparison of Bayes Blocks and BUGS software libraries would reveal strengths and weaknesses of different posterior approximations.

In control or decision making, sometimes the best decision is to first gather more information to be able to make better decisions later. This is known as probing or exploration, depending on whether information is gathered about the state of the world or the model of the world. It would be interesting to continue work by Bar-Shalom (1981) studying probing in control and by Thrun (1992) studying exploration in control.

There are many ways to combine neural (nonlinear) and logical (relational) methods. In the models presented here, the logical part defines the structure where the neural part then operates. It would be possible to let the neural part decide which logical structures to study. Such a system would be able to use computational resources more efficiently. For instance in the game of Go, a neural pattern recognition system could decide with which settings a search for local move sequences should be performed.

Bibliography

Anderberg, M. (1973). *Cluster Analysis for Applications*. Academic Press, New York, NY.

Anderson, B. and Moore, J. (1979). *Optimal Filtering*. Prentice-Hall, Englewood Cliffs, NJ.

Anderson, C., Domingos, P., and Weld, D. (2002). Relational Markov models and their application to adaptive web navigation. In Hand, D., Keim, D., Zaïne, O., and Goebel, R., editors, *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 143–152, Edmonton, Canada. ACM Press.

Attias, H. (1999). Independent factor analysis. *Neural Computation*, 11(4):803–851.

Attias, H. (2001). ICA, graphical models and variational methods. In Roberts, S. and Everson, R., editors, *Independent Component Analysis: Principles and Practice*, pages 95–112. Cambridge University Press.

Attias, H. (2003). Planning by probabilistic inference. In Bishop, C. M. and Frey, B. J., editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics (AISTATS 2003)*, Key West, Florida.

Bar-Shalom, Y. (1981). Stochastic dynamic programming: Caution and probing. *IEEE Transactions on Automatic Control*, 26(5):1184–1195.

Barber, D. and Bishop, C. M. (1998). Ensemble learning in Bayesian neural networks. In Bishop, C. M., editor, *Neural Networks and Machine Learning*, pages 215–237. Springer, Berlin.

Bayes, T. (1763/1958). Studies in the history of probability and statistics: IX. Thomas Bayes's essay towards solving a problem in the doctrine of chances. *Biometrika*, 45:296–315.

Beal, M. and Ghahramani, Z. (2003). The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures. *Bayesian Statistics 7*, 7:453–464.

Bernardo, J. M. and Smith, A. F. M. (2000). *Bayesian Theory*. J. Wiley.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press.

Bishop, C. M. (1999). Latent variable models. In Jordan, M., editor, *Learning in Graphical Models*, pages 371–403. The MIT Press, Cambridge, MA, USA.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Bromberg, F., Margaritis, D., and Honavar, V. (2006). Efficient Markov network structure discovery from independence tests. In *SIAM Data Mining 2006 (SDM06)*. To appear.

Charniak, E. (1993). *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts.

Chen, C., editor (1990). *Neural Networks For Pattern Recognition And Their Applications*. World Scientific Publishing, Singapore.

Chen, C. (1999). *Linear System Theory and Design*. Oxford University Press, Oxford. 3rd Edition.

Choudrey, R., Penny, W., and Roberts, S. (2000). An ensemble learning approach to independent component analysis. In *Proc. of the IEEE Workshop on Neural Networks for Signal Processing, Sydney, Australia, December 2000*, pages 435–444. IEEE Press.

Chui, C. and Chen, G. (1991). *Kalman Filtering: With Real-Time Applications*. Springer.

Codd, E. (1970). A relational model of data for large shared data banks. *Communications of the Association of Computing Machinery*, 13(6):377–387.

Comon, P. (1994). Independent component analysis – a new concept? *Signal Processing*, 36:287–314.

Cowell, R. G., Dawid, A. P., Lauritzen, S. L., and Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York.

Cox, R. T. (1946). Probability, frequency and reasonable expectation. *American Journal of Physics*, 14(1):1–13.

Davison, B. and Hirsh, H. (1998). Predicting sequences of user actions. In *Predicting the Future: AI Approaches to Time-Series Analysis*, pages 5–12. AAAI Press. Proceedings of AAAI-98/ICML-98 Workshop, published as Technical Report WS-98-07.

De Raedt, L., editor (1996). *Advances in Inductive Logic Programming*. IOS Press.

De Raedt, L. (2005). *From Inductive Logic Programming to Multi-Relational Data Mining*. Cognitive Technologies. Springer-Verlag.

De Raedt, L. and Kersting, K. (2003). Probabilistic Logic Learning. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, 5(1):31–48.

Dean, T. L. and Wellman, M. P. (1991). *Planning and Control*. Morgan Kaufmann.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Statistical Society, Series B (Methodological)*, 39(1):1–38.

Diez, F. (1993). Parameter adjustment in Bayes networks: The generalized noisy or-gate. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence (UAI '93)*, pages 99–105, San Francisco, CA. Morgan Kaufmann.

Doucet, A., de Freitas, N., and Gordon, N. J. (2001). *Sequential Monte Carlo Methods in Practice*. Springer Verlag.

Doyle, J. C., Francis, B. A., and Tannenbaum, A. R. (1992). *Feedback control theory*. MacMillan, New York.

Dubois, D. and Prade, H. (1993). Fuzzy sets and probability: misunderstandings, bridges and gaps. In *Proceedings of the Second IEEE Conference on Fuzzy Systems*, pages 1059–1068.

Dzeroski, S. and Lavrac, N. (2001). Introduction to inductive logic programming. In Dzeroski, S. and Lavrac, N., editors, *Relational Data Mining*, pages 48–73. Springer-Verlag.

Eduardo Fernández Camacho, C. B. (2004). *Model Predictive Control*. Springer.

Engle, R. F. and Watson, M. W. (1987). The Kalman filter: applications to forecasting and rational-expectations models. In Bewley, T. F., editor, *Advances in Econometrics Fifth World Congress*. Cambridge University Press.

Fischer, I. and Meinl, T. (2004). Graph based molecular data mining—an overview. In Thissen, W., Wieringa, P., Pantic, M., and Ludema, M., editors, *IEEE SMC 2004 Conference Proceedings*, pages 4578–4582, Den Haag, The Netherlands.

Frasconi, P., Soda, G., and Vullo, A. (2002). Hidden Markov models for text categorization in multi-page documents. *Journal of Intelligent Information Systems*, 18(2/3):195–217.

Frey, B. J. and Hinton, G. E. (1999). Variational learning in nonlinear Gaussian belief networks. *Neural Computation*, 11(1):193–214.

Friedman, N. (1997). Learning belief networks in the presence of missing values and hidden variables. In Fisher, D., editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-1997)*, pages 125–133, Nashville, Tennessee, USA. Morgan Kaufmann.

Friedman, N. (1998). The Bayesian structural EM algorithm. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'98)*, pages 129–138.

Furukawa, K., Michie, D., and Muggleton, S. (1999). *Machine Intelligence 15: Machine intelligence and inductive learning*. Oxford University Press.

Gelman, A., Carlin, J., Stern, H., and Rubin, D. (1995). *Bayesian Data Analysis*. Chapman & Hall/CRC Press, Boca Raton, Florida.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741.

Getoor, L. (2001). *Learning Statistical Models from Relational Data*. PhD thesis, Stanford University.

Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. (2001). Learning probabilistic relational models. In Džeroski, S. and Lavrač, N., editors, *Relational Data Mining*, pages 307–333. Springer-Verlag.

Getoor, L., Friedman, N., Koller, D., and Taskar, B. (2002). Learning probabilistic models of link structure. *Journal of Machine Learning Research*, 3:679–707.

Ghahramani, Z. (1998). Learning dynamic Bayesian networks. In Giles, C. and Gori, M., editors, *Adaptive Processing of Sequences and Data Structures*, Lecture Notes in Computer Science, pages 168–197. Springer-Verlag, Berlin.

Ghahramani, Z. and Beal, M. (2001). Propagation algorithms for variational Bayesian learning. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 507–513. The MIT Press, Cambridge, MA, USA.

Ghahramani, Z. and Jordan, M. (1997). Factorial hidden Markov models. *Machine Learning*, 29:245–273.

Giarratano, J. and Riley, G. (1994). *Expert Systems, Principles and Programming.* PWS Publishing Company, Boston.

Gödel, K. (1929). *Über die Vollständigkeit des Logikkalküls.* PhD thesis, University Of Vienna.

Green, P., Barker, J., Cooke, M., and Josifovski, L. (2001). Handling missing and unreliable information in speech recognition. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics (AISTATS 2001)*, pages 49–56, Key West, Florida, USA.

Hanson, C. W. and Marshall, B. (2001). Artificial intelligence applications in the intensive care unit. *Critical Care Medicine*, 29(2):427–435.

Harman, H. (1967). *Modern Factor Analysis.* University of Chicago Press, 2nd edition.

Harva, M. and Kabán, A. (2005). A variational Bayesian method for rectified factor analysis. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN'05)*, pages 185–190, Montreal, Canada.

Harva, M., Raiko, T., Honkela, A., Valpola, H., and Karhunen, J. (2005). Bayes Blocks: An implementation of the variational Bayesian building blocks framework. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005)*, pages 259–266, Edinburgh, Scotland.

Haykin, S. (1999). *Neural Networks – A Comprehensive Foundation, 2nd ed.* Prentice-Hall.

Helma, C., Gottmann, E., and Kramer, S. (2000). Knowledge discovery and data mining in toxicology. *Statistical Methods in Medical Research*, 9:329–358. Special issue on Data Mining in Medicine.

Hinton, G. E. and van Camp, D. (1993). Keeping neural networks simple by minimizing the description length of the weights. In *Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*, pages 5–13, Santa Cruz, CA, USA.

Hofmann, R. and Tresp, V. (1996). Discovering structure in continuous variables using Bayesian networks. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 500–506. The MIT Press.

Hofmann, R. and Tresp, V. (1998). Nonlinear Markov networks for continuous variables. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Advances in Neural Information Processing Systems*, volume 10, pages 521–529. The MIT Press.

Honkela, A., Harmeling, S., Lundqvist, L., and Valpola, H. (2004). Using kernel PCA for initialisation of variational Bayesian nonlinear blind source separation method. In Puntonet, C. G. and Prieto, A., editors, *Proc. of the Fifth International Conference on Independent Component Analysis and Blind Signal Separation (ICA 2004)*, volume 3195 of *Lecture Notes in Computer Science*, pages 790–797, Granada, Spain. Springer-Verlag, Berlin.

Honkela, A., Östman, T., and Vigário, R. (2005). Empirical evidence of the linear nature of magnetoencephalograms. In *Proc. 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, pages 285–290, Bruges, Belgium.

Honkela, A. and Valpola, H. (2004). Variational learning and bits-back coding: an information-theoretic view to Bayesian learning. *IEEE Transactions on Neural Networks*, 15(4):800–810.

Honkela, A. and Valpola, H. (2005). Unsupervised variational Bayesian learning of nonlinear models. In Saul, L., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 593–600. MIT Press, Cambridge, MA, USA.

Honkela, A., Valpola, H., and Karhunen, J. (2003). Accelerating cyclic update algorithms for parameter estimation by pattern searches. *Neural Processing Letters*, 17(2):191–203.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Horváth, T., Wrobel, S., and Bohnebeck, U. (2001). Relational instance-based learning with lists and terms. *Machine Learning*, 43:53–80.

Hyvärinen, A., Karhunen, J., and Oja, E. (2001). *Independent Component Analysis*. J. Wiley.

Ilin, A. and Honkela, A. (2004). Postnonlinear independent component analysis by variational Bayesian learning. In Puntonet, C. G. and Prieto, A., editors, *Proc. of the Fifth International Conference on Independent Component Analysis and Blind Signal Separation (ICA 2004)*, volume 3195 of *Lecture Notes in Computer Science*, pages 766–773, Granada, Spain. Springer-Verlag, Berlin.

Ilin, A. and Valpola, H. (2005). On the effect of the form of the posterior approximation in variational learning of ICA models. *Neural Processing Letters*, 22(2):183–204.

Ilin, A., Valpola, H., and Oja, E. (2004). Nonlinear dynamical factor analysis for state change detection. *IEEE Transactions on Neural Networks*, 15(3):559–575.

Jacobs, N. and Blockeel, H. (2001). The learning shell: Automated macro construction. In *User Modeling 2001*, pages 34–43.

Jaynes, E. T. (2003). *Probability Theory: The Logic of Science*. Cambridge University Press, Cambridge, UK.

Jensen, F., Lauritzen, S. L., and Olesen, K. G. (1990). Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, 4:269–282.

Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer-Verlag.

Jordan, M., editor (1999). *Learning in Graphical Models*. The MIT Press, Cambridge, MA, USA.

Jordan, M., Ghahramani, Z., Jaakkola, T., and Saul, L. (1999). An introduction to variational methods for graphical models. In Jordan, M., editor, *Learning in Graphical Models*, pages 105–161. The MIT Press, Cambridge, MA, USA.

Julier, S. and Uhlmann, J. (1997). A new extension of the Kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*.

Jutten, C. and Karhunen, J. (2004). Advances in blind source separation (BSS) and independent component analysis (ICA) for nonlinear mixtures. *International Journal of Neural Systems*, 14(5):267–292.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

Kendall, M. (1975). *Multivariate Analysis*. Charles Griffin & Co.

Kersting, K. and De Raedt, L. (2001). Bayesian logic programs. Technical Report 151, Institute for Computer Science, University of Freiburg, Germany.

Kersting, K. and De Raedt, L. (2006). Bayesian Logic Programming: Theory and tool. In Getoor, L. and Taskar, B., editors, *An Introduction to Statistical Relational Learning*. MIT Press. To appear.

Kersting, K. and Landwehr, N. (2004). Scaled conjugate gradients for maximum likelihood: An empirical comparison with the EM algorithm. In J. A. Gámez, S. M. and Salmerón, A., editors, *"Advances in Bayesian Networks", Series: Studies in Fuzziness and Soft Computing*, volume 146, pages 235–254. Springer.

Kirk, D. E. (2004). *Optimal Control Theory*. Courier Dover Publications.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Klir, G. and Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice-Hall Inc.

Kohonen, T. (2001). *Self-Organizing Maps*. Springer, 3rd, extended edition.

Koller, D. (1999). Probabilistic relational models. In Džeroski, S. and Flach, P., editors, *Proceedings of Ninth International Workshop on Inductive Logic Programming (ILP-99)*, volume 1634 of *LNAI*, pages 3–13, Bled, Slovenia. Springer.

Korvemaker, B. and Greiner, R. (2000). Predicting UNIX command files: Adjusting to user patterns. In *Adaptive User Interfaces: Papers from the 2000 AAAI Spring Symposium*, pages 59–64.

Koski, T. (2001). *Hidden Markov Models for Bioinformatics*. Kluwer Academic Publishers.

Landwehr, N., Kersting, K., and De Raedt, L. (2005). nFOIL: Integrating Naïve Bayes and Foil. In Veloso, M. and Kambhampat, S., editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 275–282, Pittsburgh, Pennsylvania, USA. AAAI Press.

Landwehr, N., Mielikäinen, T., Eronen, L., Toivonen, H., and Mannila, H. (2006). Constrained hidden markov models for population-based haplotyping. In Rouso, J., Kaski, S., and Ukkonen, E., editors, *Proceedings of the Workshop on Probabilistic Modeling and Machine Learning in Structural and Systems Biology (PMSB)*, Tuusula, Finland.

Lane, T. (1999). Hidden Markov models for human/computer interface modeling. In Rudström, Å., editor, *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pages 35–44, Stockholm, Sweden.

Lappalainen, H. and Honkela, A. (2000). Bayesian nonlinear independent component analysis by multi-layer perceptrons. In Girolami, M., editor, *Advances in Independent Component Analysis*, pages 93–121. Springer-Verlag, Berlin.

Lappalainen, H. and Miskin, J. (2000). Ensemble learning. In Girolami, M., editor, *Advances in Independent Component Analysis*, pages 75–92. Springer-Verlag, Berlin.

Lasserre, J., Bishop, C. M., and Minka, T. (2006). Principled hybrids of generative and discriminative models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, New York.

Lavrac, N. and Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York.

Little, R. and D.B.Rubin (1987). *Statistical Analysis with Missing Data*. J. Wiley & Sons.

Lloyd, J. (2003). *Logic for Learning: Learning Comprehensible Theories from Structured Data*. Springer-Verlag.

MacKay, D. J. C. (1995a). Developments in probabilistic modelling with neural networks – ensemble learning. In *Neural Networks: Artificial Intelligence and Industrial Applications. Proc. of the 3rd Annual Symposium on Neural Networks*, pages 191–198.

MacKay, D. J. C. (1995b). Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6:469–505.

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

Maybeck, P. S. (1979). *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press.

Meila, M. and Jordan, M. I. (1996). Learning fine motion by markov mixtures of experts. In Touretzky, D., Mozer, M. C., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems 8*. MIT Press.

Meng, X. L. and van Dyk, D. A. (1995). Augmenting data wisely to speed up the em algorithm. In *Proceedings of the Statistical Computing Section of the American Statistical Association*, pages 160–165.

Minka, T. (2001). Expectation propagation for approximate Bayesian inference. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, UAI 2001*, pages 362–369.

Miskin, J. and MacKay, D. J. C. (2001). Ensemble learning for blind source separation. In Roberts, S. and Everson, R., editors, *Independent Component Analysis: Principles and Practice*, pages 209–233. Cambridge University Press.

Morari, M. and Lee, J. (1999). Model predictive control: Past, present and future. *Computers and Chemical Engineering*, pages 667–682.

Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing Journal*, 13:245–286.

Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679.

Muggleton, S. and Feng, C. (1992). Efficient induction in logic programs. In Muggleton, S., editor, *Inductive Logic Programming*, pages 281–298. Academic Press.

Murphy, K. P. (2001). An introduction to graphical models. Technical report, Intel Research.

Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 467–475.

Murzin, A. G., Brenner, S. E., Hubbard, T., and Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

Neal, R. M. (2001). Annealed importance sampling. *Statistics and Computing*, 11(2):125–139.

Neal, R. M. and Hinton, G. E. (1999). A view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan, M. I., editor, *Learning in Graphical Models*, pages 355–368. The MIT Press, Cambridge, MA, USA.

Neapolitan, R. E. (2004). *Learning Bayesian Networks.* Pearson Prentice Hall, Upper Saddle River, NJ.

Nolan, L., Harva, M., Kabán, A., and Raychaudhury, S. (2006). A data-driven Bayesian approach for finding young stellar populations in early-type galaxies from their UV-optical spectra. *Monthly Notices of the Royal Astronomical Society*, 366(1):321–338.

Palomäki, K. J., Brown, G. J., and Barker, J. (2004). Techniques for handling convolutional distortion with "missing data" automatic speech recognition. *Speech Communication*, 43:123–142.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufman, San Francisco.

Petersen, K. B., Winther, O., and Hansen, L. K. (2005). On the slow convergence of EM and VBEM in low noise linear mixtures. *Neural Computation*, 17(9):1921–1926.

Pietraszek, T. and Tanner, A. (2005). Data mining and machine learning—towards reducing false positives in intrusion detection. *Information Security Technical Report Journal*, 10(3):169–183.

Popescul, A., Ungar, L., Lawrence, S., and Pennock, D. (2003). Statistical relational learning for document mining. In *Proceedings of the IEEE International Conference on Data Mining (ICDM-03)*, pages 275–282.

Psiaki, M. (2005). Backward-smoothing extended Kalman filter. *Journal of Guidance, Control, and Dynamics*, 28(5).

Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3):239–266.

Rabiner, L. R. and Juang, B. H. (1986). An introduction to hidden Markov models. *IEEE Acoustics, Speech, and Signal Processing Magazine*, 3(1):4–15.

Raiko, T. (2001). Hierarchical nonlinear factor analysis. Master's thesis, Helsinki University of Technology, Espoo.

Raiko, T., Kersting, K., Karhunen, J., and De Raedt, L. (2002). Bayesian learning of logical hidden markov models. In *Proceedings of the Finnish Artificial Intelligence Conference (STeP 2002)*, pages 64–71, Oulu, Finland.

Raju, K., Ristaniemi, T., Karhunen, J., and Oja, E. (2006). Jammer suppression in DS-CDMA arrays using independent component analysis. *IEEE Trans. on Wireless Communications*, 5(1):77–82.

Reiter, R. (1978). On closed world data bases. In *Logic and Data Bases*, pages 119–140. Plenum Publ. Co., New York.

Resnik, M. (1987). *Choices: An Introduction to Decision Theory*. University of Minnesota Press, Minneapolis, Minnesota.

Ristaniemi, T. (2000). *Synchronization and Blind Signal Processing in CDMA Systems*. PhD thesis, University of Jyväskylä, Jyväskylä, Finland.

Ristic, B., Arulampalam, S., and Gordon, N. (2004). *Beyond the Kalman Filter*. Artech House.

Roberts, S. and Everson, R. (2001). Introduction. In Roberts, S. and Everson, R., editors, *Independent Component Analysis: Principles and Practice*, pages 1–70. Cambridge University Press.

Rosenqvist, F. and Karlström, A. (2005). Realisation and estimation of piecewise-linear output-error models. *Automatica*, 41(3):545–551.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence A Modern Approach*. Prentice-Hall, New Jersey.

Salakhutdinov, R., Roweis, S. T., and Ghahramani, Z. (2003). Optimization with EM and expectation-conjugate-gradient. In *Proceedings of the international conference on machine learning (ICML-2003)*, pages 672–679.

Särkkä, S., Vehtari, A., and Lampinen, J. (2006). Rao-Blackwellized particle filter for multiple target tracking. *Information Fusion*. to appear.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464.

Segal, E., Taskar, B., Gasch, A., Friedman, N., and Koller, D. (2001). Rich probabilistic models for gene expression. *Bioinformatics*, 17:243–252.

Seltzer, M., Raj, B., and Stern, R. (2004). A Bayesian framework for spectrographic mask estimation for missing feature speech recognition. *Speech Communication*, 43(4):379–393.

Spiegelhalter, D., Thomas, A., Best, N., and Gilks, W. (1995). BUGS: Bayesian inference using Gibbs sampling, version 0.50.

Srinivasan, A. (2005). The Aleph manual. Available at `http://web.comlab.ox.ac.uk/oucl/work/ashwin.srinivasan/`.

Sterling, L. and Shapiro, E. (1994). *The Art of Prolog*. The MIT Press, second edition.

Stinchcombe, M. and White, H. (1989). Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN '89)*, pages I–613–617.

Stone, M. (1974). Cross-validation choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147.

Taskar, B., Abbeel, P., and Koller, D. (2002). Discriminative probabilistic models for relational data. In *Proc. Conference on Uncertainty in Artificial Intelligence (UAI02)*, pages 485–492, Edmonton.

Thrun, S. (1992). The role of exploration in learning control. In White, D. and Sofge, D., editors, *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, Florence, Kentucky 41022.

Tornio, M. and Raiko, T. (2006). Variational Bayesian approach for nonlinear identification and control. In *Proceedings of the IFAC Workshop on Nonlinear Model Predictive Control for Fast Systems (NMPC FS06)*, Grenoble, France. To appear.

Valpola, H., Harva, M., and Karhunen, J. (2004). Hierarchical models of variance sources. *Signal Processing*, 84(2):267–282.

Valpola, H., Honkela, A., Harva, M., Ilin, A., Raiko, T., and Östman, T. (2003a). Bayes blocks software library. Available at http://www.cis.hut.fi/projects/bayes/software/.

Valpola, H. and Karhunen, J. (2002). An unsupervised ensemble learning method for nonlinear dynamic state-space models. *Neural Computation*, 14(11):2647–2692.

Valpola, H., Östman, T., and Karhunen, J. (2003b). Nonlinear independent factor analysis by hierarchical models. In *Proc. 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*, pages 257–262, Nara, Japan.

Valpola, H., Raiko, T., and Karhunen, J. (2001). Building blocks for hierarchical latent variable models. In *Proc. 3rd Int. Conf. on Independent Component Analysis and Signal Separation (ICA2001)*, pages 710–715, San Diego, USA.

Vigário, R., Jousmäki, V., Hämäläinen, M., Hari, R., and Oja, E. (1998). Independent component analysis for identification of artifacts in magnetoencephalographic recordings. In *Advances in Neural Information Processing System 10 (Proc. NIPS 97)*, pages 229–235. MIT Press.

Wallace, C. S. (1990). Classification by minimum-message-length inference. In Aki, S. G., Fiala, F., and Koczkodaj, W. W., editors, *Advances in Computing and Information – ICCI '90*, volume 468 of *Lecture Notes in Computer Science*, pages 72–81. Springer, Berlin.

Winn, J. and Bishop, C. M. (2005). Variational message passing. *Journal of Machine Learning Research*, 6:661–694.

Winn, J. and Joijic, N. (2005). LOCUS: Learning object classes with unsupervised segmentation. In *Proc. IEEE Intl. Conf. on Computer Vision (ICCV)*, pages 756–763, Beijing.

Won, K.-J., Prugel-Bennett, A., and Krogh, A. (2006). Evolving the structure of hidden Markov models. *IEEE Transactions on Evolutionary Computation*, 10(1):39–49.

Publication 1

T. Raiko, H. Valpola, M. Harva, and J. Karhunen. Building Blocks for Variational Bayesian Learning of Latent Variable Models. Report E4 in the Electronic report series of CIS, April, 2006, accepted for publication conditioned on minor revisions to the *Journal of Machine Learning Research*.

Building Blocks for Variational Bayesian Learning of Latent Variable Models

Tapani Raiko, Harri Valpola, Markus Harva, and Juha Karhunen
Helsinki University of Technology, Adaptive Informatics Research Centre
P.O.Box 5400, FI-02015 HUT, Espoo, FINLAND
email: `firstname.lastname@tkk.fi`
URL: `http://www.cis.hut.fi/projects/bayes/`
Fax: +358-9-451 3277

April 26, 2006

Abstract

We introduce standardised building blocks designed to be used with variational Bayesian learning. The blocks include Gaussian variables, summation, multiplication, nonlinearity, and delay. A large variety of latent variable models can be constructed from these blocks, including variance models and nonlinear modelling, which are lacking from most existing variational systems. The introduced blocks are designed to fit together and to yield efficient update rules. Practical implementation of various models is easy thanks to an associated software package which derives the learning formulas automatically once a specific model structure has been fixed. Variational Bayesian learning provides a cost function which is used both for updating the variables of the model and for optimising the model structure. All the computations can be carried out locally, resulting in linear computational complexity. We present experimental results on several structures, including a new hierarchical nonlinear model for variances and means. The test results demonstrate the good performance and usefulness of the introduced method.

1 Introduction

Various generative modelling approaches have provided powerful statistical learning methods for neural networks and graphical models during the last years. Such methods aim at finding an appropriate model which explains the internal structure or regularities found in the observations. It is assumed that these regularities are caused by certain latent variables (also called factors, sources, hidden variables, or hidden causes) which have generated the observed data through an unknown mapping [9]. In unsupervised learning, the goal is to identify both the unknown latent variables and generative mapping, while in supervised learning it suffices to estimate the generative mapping.

The expectation-maximisation (EM) algorithm has often been used for learning latent variable models [8, 9, 39, 41]. The distribution for the latent variables is modelled, but the model parameters are found using maximum likelihood or maximum a posteriori estimators. However, with such point estimates, determination of the correct model order and overfitting are ubiquitous and often difficult problems. Therefore, full Bayesian approaches making use of the complete posterior distribution have recently gained a lot of attention. Exact treatment of the posterior distribution is intractable except in simple toy problems, and hence one must resort to suitable approximations. So-called Laplacian approximation method [46, 8] employs a Gaussian approximation around the peak of the posterior distribution. However, this method still suffers from overfitting. In real-world problems, it often does not perform adequately, and has therefore largely given way for better alternatives. Among them, Markov Chain Monte Carlo (MCMC) techniques [49, 51, 56] are popular in supervised learning tasks, providing good estimation results. Unfortunately, the computational load is high, which restricts the use of MCMC in large scale unsupervised learning problems where the parameters and variables to be estimated are numerous. For instance, [68] has a case study in unsupervised learning from brain imaging data. He used MCMC for a scaled down toy example but resorted to point estimates with real data.

Ensemble learning [26, 47, 51, 5, 45], which is one of the variational Bayesian methods [40, 3, 41], has gained increasing attention during the last years. This is because it largely avoids overfitting, allows for estimation of the model order and structure, and its computational load is reasonable compared to the MCMC methods. Variational Bayesian learning was first employed in supervised problems [80, 26, 47, 5], but it has now become popular also in unsupervised modelling. Recently, several authors have successfully applied such techniques to linear factor analysis, independent component analysis (ICA) [36, 66, 12, 27], and their various extensions. These include linear independent factor analysis [2], several other extensions of the basic linear ICA model [4, 11, 53, 67], as well as MLP networks for modelling nonlinear observation mappings [44, 36] and nonlinear dynamics of the latent variables (source signals) [38, 74, 75]. Variational Bayesian learning has also been applied to large discrete models [54] such as nonlinear belief networks [15] and hidden Markov models [48].

In this paper, we introduce a small number of basic blocks for building latent variable models which are learned using variational Bayesian learning. The blocks have been introduced earlier in two conference papers [77, 23] and their applications in [76, 33, 30, 65, 62, 63]. [71] studied hierarchical models for variance sources from signal-processing point of view. This paper is the first comprehensive presentation about the block framework itself. Our approach is most suitable for unsupervised learning tasks which are considered in this paper, but in principle at least, it could be applied to supervised learning, too. A wide variety of factor-analysis-type latent-variable models can be constructed by combining the basic blocks suitably. Variational Bayesian learning then provides a cost function which can be used for updating the variables as well as for optimising the model structure. The blocks are designed so as to fit together and yield efficient update rules. By using a maximally factorial posterior approximation, all the required computations can be performed locally. This results in linear computational complexity as a function of the number of connections in the model. The Bayes Blocks software package by [72] is an open-source C++/Python implementation that can freely be downloaded.

The basic building block is a Gaussian variable (node). It uses as its input values both mean and variance. The other building blocks include addition and multiplication nodes, delay, and a Gaussian variable followed by a nonlinearity. Several known model structures can be constructed using these blocks. We also introduce some novel model structures by extending known linear structures using nonlinearities and variance modelling. Examples will be presented later on in this paper.

The key idea behind developing these blocks is that after the connections between the blocks in the chosen model have been fixed (that is, a particular model has been selected and specified), the cost function and the updating rules needed in learning can be computed automatically. The user does not need to understand the underlying mathematics since the derivations are done within the software package. This allows for rapid prototyping. The Bayes Blocks can also be used to bring different methods into a unified framework, by implementing a corresponding structure from blocks and by using results of these methods for initialisation. Different methods can then be compared directly using the cost function and perhaps combined to find even better models. Updates that minimise a global cost function are guaranteed to converge, unlike algorithms such as loopy belief propagation [60], extended Kalman smoothing [1], or expectation propagation [52].

[81] have introduced a general purpose algorithm called variational message passing. It resembles our framework in that it uses variational Bayesian learning and factorised approximations. The VIBES framework allows for discrete variables but not nonlinearities or nonstationary variance. The posterior approximation does not need to be fully factorised which leads to a more accurate model. Optimisation proceeds by cycling through each factor and revising the approximate posterior distribution. Messages that contain certain expectations over the posterior approximation are sent through the network.

[7, 17], and [6] view variational Bayesian learning as an extension to

the EM algorithm. Their algorithms apply to combinations of discrete and linear Gaussian models. In the experiments, the variational Bayesian model structure selection outperformed the Bayesian information criterion [69] at relatively small computational cost, while being more reliable than annealed importance sampling even with the number of samples so high that the computational cost is hundredfold.

A major difference of our approach compared to the related methods by [81] and by [7] is that they concentrate mainly on situations where there is a handy conjugate prior [16] of the posterior distributions available. This makes life easier, but on the other hand our blocks can be combined more freely, allowing richer model structures. For instance, the modelling of variance in a way described in Section 5.1, would not be possible using the gamma distribution for the precision parameter in the Gaussian node. The price we have to pay for this advantage is that the minimum of the cost function must be found iteratively, while it can be solved analytically when conjugate distributions are applied. The cost function can always be evaluated analytically in the Bayes Blocks framework as well. Note that the different approaches would fit together.

Similar graphical models can be learned with sampling based algorithms instead of variational Bayesian learning. For instance, the BUGS software package by [70] uses Gibbs sampling for Bayesian inference. It supports mixture models, nonlinearities, and nonstationary variance. There are also many software packages concentrated on discrete Bayesian networks. Notably, the Bayes Net toolbox by [55] can be used for Bayesian learning and inference of many types of directed graphical models using several methods. It also includes decision-theoretic nodes. Hence it is in this sense more general than our work. A limitation of the Bayes net toolbox [55] is that it supports latent continuous nodes only with Gaussian or conditional Gaussian distributions.

Autobayes [20] is a system that generates code for efficient implementations of algorithms used in Bayes networks. Currently the algorithm schemas include EM, k-means, and discrete model selection. This system does not yet support continuous hidden variables, nonlinearities, variational methods, MCMC, or temporal models. One of the greatest strengths of the code generation approach compared to a software library is the possibility of automatically optimising the code using domain information.

In the independent component analysis community, traditionally, the observation noise has not been modelled in any way. Even when it is modelled, the noise variance is assumed to have a constant value which is estimated from the available observations when required. However, more flexible variance models would be highly desirable in a wide variety of situations. It is well-known that many real-world signals or data sets are nonstationary, being roughly stationary on fairly short intervals only. Quite often the amplitude level of a signal varies markedly as a function of time or position, which means that its variance is nonstationary. Examples include financial data sets, speech signals, and natural images.

Recently, [59] have demonstrated that several higher-order statistical properties of natural images and signals are well explained by a stochastic model in which an otherwise stationary Gaussian process has a nonstationary variance. Variance models are also useful in explaining volatility of financial time series and in detecting outliers in the data. By utilising the nonstationarity of variance it is possible to perform blind source separation on certain conditions [36, 61].

Several authors have introduced hierarchical models related to those discussed in this paper. These models use subspaces of dependent features instead of single feature components. This kind of models have been proposed at least in context with independent component analysis [10, 35, 34, 58], and topographic or self-organising maps [43, 18]. A problem with these methods is that it is difficult to learn the structure of the model or to compare different model structures.

The remainder of this paper is organised as follows. In the following section, we briefly present basic concepts of variational Bayesian learning. In Section 3, we introduce the building blocks (nodes), and in Section 4 we discuss variational Bayesian computations with them. In the next section, we show examples of different types of models which can be constructed using the building blocks. Section 6 deals with learning and potential problems related with it, and in Section 7 we present experimental results on several structures given in Section 5. This is followed by a short discussion as well as conclusions in the last section of the paper.

2 Variational Bayesian learning

In Bayesian data analysis and estimation methods [51, 16, 39, 56], all the uncertain quantities are modelled in terms of their joint probability density function (pdf). The key principle is to construct the joint posterior pdf for all the unknown quantities in a model, given the data sample. This posterior density contains all the relevant information on the unknown variables and parameters.

Denote by $\boldsymbol{\theta}$ the set of all model parameters and other unknown variables that we wish to estimate from a given data set $\boldsymbol{X}$. The posterior probability density $p(\boldsymbol{\theta}|\boldsymbol{X})$ of the parameters $\boldsymbol{\theta}$ given the data $\boldsymbol{X}$ is obtained from Bayes rule[1]

$$p(\boldsymbol{\theta}|\boldsymbol{X}) = \frac{p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{X})} \tag{1}$$

Here $p(\boldsymbol{X}|\boldsymbol{\theta})$ is the likelihood of the parameters $\boldsymbol{\theta}$ given the data $\boldsymbol{X}$, $p(\boldsymbol{\theta})$ is the prior pdf of these parameters, and

$$p(\boldsymbol{X}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{2}$$

is a normalising term which is called the evidence. The evidence can be directly understood as the marginal probability of the observed data $\boldsymbol{X}$ assuming the chosen model $\mathcal{H}$. By evaluating the evidences $p(\boldsymbol{X})$ for different models $\mathcal{H}_i$, one can therefore choose the model which describes the observed data with the highest probability[2] [8, 51].

Variational Bayesian learning [5, 26, 45, 47, 51] is a fairly recently introduced [26, 47] approximate fully Bayesian method, which has become popular because of its good properties. Its key idea is to approximate the exact posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{X})$ by another distribution $q(\boldsymbol{\theta})$ that is computationally easier to handle. The approximating distribution is usually chosen to be a product of several independent distributions, one for each parameter or a set of similar parameters.

Variational Bayesian learning employs the Kullback-Leibler (KL) information (divergence) between two probability distributions $q(v)$ and $p(v)$. The KL information is defined by the cost function [24]

$$\mathcal{J}_{\mathrm{KL}}(q \parallel p) = \int_v q(v) \ln \frac{q(v)}{p(v)} dv \tag{3}$$

which measures the difference in the probability mass between the densities $q(v)$ and $p(v)$. Its minimum value 0 is achieved when the densities $q(v)$ and $p(v)$ are the same.

The KL information is used to minimise the misfit between the actual posterior pdf $p(\boldsymbol{\theta}|\boldsymbol{X})$ and its parametric approximation $q(\boldsymbol{\theta})$. However, the exact KL information $\mathcal{J}_{\mathrm{KL}}(q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\boldsymbol{X}))$ between these two densities does not yet yield a practical cost function, because the normalising term $p(\boldsymbol{X})$ needed in computing $p(\boldsymbol{\theta}|\boldsymbol{X})$ cannot usually be evaluated.

Therefore, the cost function used in variational Bayesian learning is defined [26, 47]

$$\mathcal{C}_{\mathrm{KL}} = \mathcal{J}_{\mathrm{KL}}(q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\boldsymbol{X})) - \ln p(\boldsymbol{X}) \tag{4}$$

[1]The subscripts of all pdf's are assumed to be the same as their arguments, and are omitted for keeping the notation simpler.

[2]More accurately, one could show the dependence on the chosen model $\mathcal{H}$ by conditioning all the pdf's in (1) by $\mathcal{H}$: $p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H})$, $p(\boldsymbol{X}|\mathcal{H})$, etc. We have here dropped also the dependence on $\mathcal{H}$ out for notational simplicity. See [74] for a somewhat more complete discussion of Bayesian methods and ensemble learning.

After slight manipulation, this yields

$$\mathcal{C}_{\mathrm{KL}} = \int_{\boldsymbol{\theta}} q(\boldsymbol{\theta}) \ln \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{X}, \boldsymbol{\theta})} d\boldsymbol{\theta} \tag{5}$$

which does not require $p(\boldsymbol{X})$ any more. The cost function $\mathcal{C}_{\mathrm{KL}}$ consists of two parts:

$$\mathcal{C}_q = \langle \ln q(\boldsymbol{\theta}) \rangle = \int_{\boldsymbol{\theta}} q(\boldsymbol{\theta}) \ln q(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{6}$$

$$\mathcal{C}_p = \langle - \ln p(\boldsymbol{X}, \boldsymbol{\theta}) \rangle = -\int_{\boldsymbol{\theta}} q(\boldsymbol{\theta}) \ln p(\boldsymbol{X}, \boldsymbol{\theta}) d\boldsymbol{\theta} \tag{7}$$

where the shorthand notation $\langle \cdot \rangle$ denotes expectation with respect to the approximate pdf $q(\boldsymbol{\theta})$.

In addition, the cost function $\mathcal{C}_{\mathrm{KL}}$ provides a bound for the evidence $p(\boldsymbol{X})$. Since $\mathcal{J}_{\mathrm{KL}}(q \parallel p)$ is always nonnegative, it follows directly from (4) that

$$\mathcal{C}_{\mathrm{KL}} \geq - \ln p(\boldsymbol{X}) \tag{8}$$

This shows that the negative of the cost function bounds the log-evidence from below.

It is worth noting that variational Bayesian ensemble learning can be derived from information-theoretic minimum description length coding as well [26]. Further considerations on such arguments, helping to understand several common problems and certain aspects of learning, have been presented in a recent paper [31].

The dependency structure between the parameters in our method is the same as in Bayesian networks [60]. Variables are seen as nodes of a graph. Each variable is conditioned by its parents. The difficult part in the cost function is the expectation $\langle \ln p(\boldsymbol{X}, \boldsymbol{\theta}) \rangle$ which is computed over the approximation $q(\boldsymbol{\theta})$ of the posterior pdf. The logarithm splits the product of simple terms into a sum. If each of the simple terms can be computed in constant time, the overall computational complexity is linear.

In general, the computation time is constant if the parents are independent in the posterior pdf approximation $q(\boldsymbol{\theta})$. This condition is satisfied if the joint distribution of the parents in $q(\boldsymbol{\theta})$ decouples into the product of the approximate distributions of the parents. That is, each term in $q(\boldsymbol{\theta})$ depending on the parents depends only on one parent. The independence requirement is violated if any variable receives inputs from a latent variable through multiple paths or from two latent variables which are dependent in $q(\boldsymbol{\theta})$, having a non-factorisable joint distribution there. Figure 1 illustrates the flow of information in the network in these two qualitatively different cases.

Our choice for $q(\boldsymbol{\theta})$ is a multivariate Gaussian density with a diagonal covariance matrix. Even this crude approximation is adequate for finding

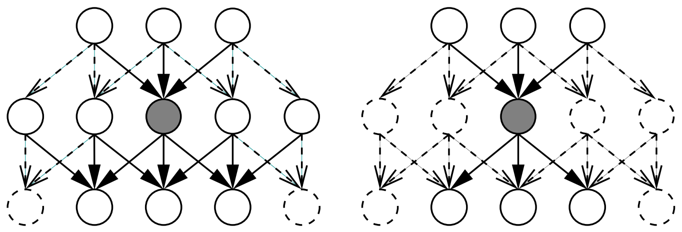


Figure 1: The dash-lined nodes and connections can be ignored while updating the shadowed node. *Left:* In general, the whole Markov blanket needs to be considered. *Right:* A completely factorial posterior approximation with no multiple computational paths leads to a decoupled problem. The nodes can be updated locally.

the region where the mass of the actual posterior density is concentrated. The mean values of the components of the Gaussian approximation provide reasonably good point estimates of the corresponding parameters or variables, and the respective variances measure the reliability of these estimates. However, occasionally the diagonal Gaussian approximation can be too crude. This problem has been considered in context with independent component analysis in [37], giving means to remedy the situation.

Taking into account posterior dependencies makes the posterior pdf approximation $q(\boldsymbol{\theta})$ more accurate, but also usually increases the computational load significantly. We have earlier considered networks with multiple computational paths in several papers, for example [44, 73, 74, 75]. The computational load of variational Bayesian learning then becomes roughly quadratically proportional to the number of unknown variables in the MLP network model used in [44, 75, 32].

The building blocks (nodes) introduced in this paper together with associated structural constraints provide effective means for combating the drawbacks mentioned above. Using them, updating at each node takes place locally with no multiple paths. As a result, the computational load scales linearly with the number of estimated quantities. The cost function and the learning formulas for the unknown quantities to be estimated can be evaluated automatically once a specific model has been selected, that is, after the connections between the blocks used in the model have been fixed. This is a very important advantage of the proposed block approach.

3 Node types

In this section, we present different types of nodes that can be easily combined together. Variational Bayesian inference algorithm for the nodes is then discussed in Section 4.

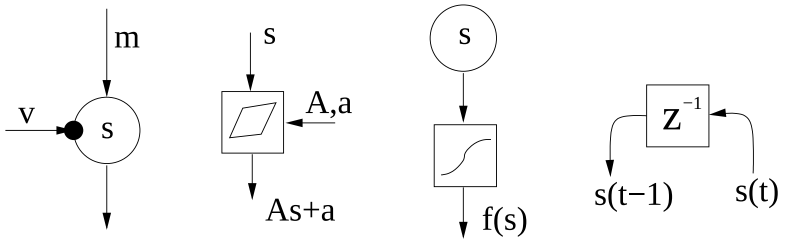


Figure 2: *First subfigure from the left:* The circle represents a Gaussian node corresponding to the latent variable s conditioned by mean m and variance $\exp(-v)$. *Second subfigure:* Addition and multiplication nodes are used to form an affine mapping from s to $As + a$. *Third subfigure:* A nonlinearity f is applied immediately after a Gaussian variable. *The rightmost subfigure:* Delay operator delays a time-dependent signal by one time unit.

In general, the building blocks can be divided into variable nodes, computation nodes, and constants. Each variable node corresponds to a random variable, and it can be either observed or hidden. In this paper we present only one type of variable node, the Gaussian node, but others can be used in the same framework. The computation nodes are the addition node, the multiplication node, a nonlinearity, and the delay node.

In the following, we shall refer to the inputs and outputs of the nodes. For a variable node, its inputs are the parameters of the conditional distribution of the variable represented by that node, and its output is the value of the variable. For computation nodes, the output is a fixed function of the inputs. The symbols used for various nodes are shown in Figure 2. Addition and multiplication nodes are not included, since they are typically combined to represent the effect of a linear transformation, which has a symbol of its own. An output signal of a node can be used as input by zero or more nodes that are called the children of that node. Constants are the only nodes that do not have inputs. The output is a fixed value determined at creation of the node.

Nodes are often structured in vectors or matrices. Assume for example that we have a data matrix $\mathbf{X} = [\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(T)]$, where $t = 1, 2, \dots T$ is called the time index of an n-dimensional observation vector. Note that t does not have to correspond to time in the real world, e.g. different t could point to different people. In the implementation, the nodes are either vectors so that the values indexed by t (e.g. observations) or scalars so that the values are constants w.r.t. t (e.g. weights). The data $\mathbf{X}$ would be represented with n vector nodes. A scalar node can be a parent of a vector node, but not a child of a vector node.

3.1 Gaussian node

The Gaussian node is a variable node and the basic element in building hierarchical models. Figure 2 (leftmost subfigure) shows the schematic diagram of the Gaussian node. Its output is the value of a Gaussian random variable s, which is conditioned by the inputs m and v. Denote generally by $\mathcal{N}(x; m_x, \sigma_x^2)$ the probability density function of a Gaussian random variable x having the mean m_x and variance σ_x^2. Then the conditional probability function (cpf) of the variable s is $p(s \mid m, v) = \mathcal{N}(s; m, \exp(-v))$. As a generative model, the Gaussian node takes its mean input m and adds to it Gaussian noise (or innovation) with variance $\exp(-v)$.

Variables can be latent or observed. Observing a variable means fixing its output s to the value in the data. Section 4 is devoted to inferring the distribution over the latent variables given the observed variables. Inferring the distribution over variables that are independent of t is also called learning.

3.2 Computation nodes

The addition and multiplication nodes are used for summing and multiplying variables. These standard mathematical operations are typically used to construct linear mappings between the variables. This task is automated in the software, but in general, the nodes can be connected in other ways, too. An addition node that has n inputs denoted by $s_1, s_2, \ldots, s_n$, gives the sum of its inputs as the output $\sum_{i=1}^n s_i$. Similarly, the output of a multiplication node is the product of its inputs $\prod_{i=1}^n s_i$.

A nonlinear computation node can be used for constructing nonlinear mappings between the variable nodes. The nonlinearity

$$f(s) = \exp(-s^2) \tag{9}$$

is chosen because the required expectations can be solved analytically for it. Another implemented nonlinearity for which the computations can be carried out analytically is the cut function $g(s) = \max(s, 0)$. Other possible nonlinearities are discussed in Section 4.3.

3.3 Delay node

The delay operation can be used to model dynamics. The node operates on time-dependent signals. It transforms the inputs $s(1), s(2), \ldots, s(T)$ into outputs $s_0, s(1), s(2), \ldots, s(T-1)$ where s_0 is a scalar parameter that provides a starting distribution for the dynamical process. The symbol z^{-1} in the rightmost subfigure of Fig. 2 illustrating the delay node is the standard notation for the unit delay in signal processing and temporal neural networks [24]. Models containing the delay node are called dynamic, and the other models are called static.

4 Variational Bayesian inference in Bayes blocks

In this section we give equations needed for computation with the nodes introduced in Section 3. Generally speaking, each node propagates to the forward direction a distribution of its output given its inputs. In the backward direction, the dependency of the cost function (5) of the children on the output of their parent is propagated. These two potentials are combined to form the posterior distribution of each variable. There is a direct analogy to Bayes rule (1): the prior (forward) and the likelihood (backward) are combined to form the posterior distribution. We will show later on that the potentials in the two directions are fully determined by a few values, which consist of certain expectations over the distribution in the forward direction, and of gradients of the cost function w.r.t. the same expectations in the backward direction.

In the following, we discuss in more detail the properties of each node. Note that the delay node does actually not process the signals, it just rewires them. Therefore no formulas are needed for its associated the expectations and gradients.

4.1 Gaussian node

Recall the Gaussian node in Section 3.1. The variance is parameterised using the exponential function as $\exp(-v)$. This is because then the mean $\langle v \rangle$ and expected exponential $\langle \exp v \rangle$ of the input v suffice for evaluating the cost function, as will be shown shortly. Consequently the cost function can be minimised using the gradients with respect to these expectations. The gradients are computed backwards from the children nodes, but otherwise our learning method differs clearly from standard back-propagation [24].

Another important reason for using the parameterisation $\exp(-v)$ for the prior variance of a Gaussian random variable s is that the posterior distribution of s then becomes approximately Gaussian, provided that the prior mean m of s is Gaussian, too (see for example Section 7.1 or [45]). The conjugate prior distribution of the inverse of the prior variance of a Gaussian random variable is the gamma distribution [16]. Using such gamma prior pdf causes the posterior distribution to be gamma, too, which is mathematically convenient. However, the conjugate prior pdf of the second parameter of the gamma distribution is something quite intractable. Hence gamma distribution is not suitable for developing hierarchical variance models. The logarithm of a gamma distributed variable is approximately Gaussian distributed [16], justifying the adopted parameterisation $\exp(-v)$. However, it should be noted that both the gamma and $\exp(-v)$ distributions are used as prior pdfs mainly because they make the estimation of the posterior pdf mathematically tractable [45]; one cannot claim that either of these choices would be correct.

4.1.1 Cost function

Recall now that we are approximating the joint posterior pdf of the random variables s, m, and v in a maximally factorial manner. It then decouples into the product of the individual distributions: $q(s, m, v) = q(s)q(m)q(v)$. Hence s, m, and v are assumed to be statistically independent a posteriori. The posterior approximation $q(s)$ of the Gaussian variable s is defined to be Gaussian with mean $\overline{s}$ and variance $\widetilde{s}$: $q(s) = \mathcal{N}(s; \overline{s}, \widetilde{s})$. Utilising these, the part $\mathcal{C}_p$ of the Kullback-Leibler cost function arising from the data, defined in Eq. (7), can be computed in closed form. For the Gaussian node of Figure 2, the cost becomes

$$\begin{aligned}\mathcal{C}_{s,p} &= -\left\langle \ln p(s|m,v) \right\rangle \\ &= \frac{1}{2}\Big\{ \left\langle \exp v \right\rangle \Big[(\overline{s} - \langle m \rangle)^2 + \operatorname{Var}\{m\} + \widetilde{s} \Big] - \langle v \rangle + \ln 2\pi \Big\} \end{aligned} \tag{10}$$

The derivation is presented in Appendix B of [74] using slightly different notation. For the observed variables, this is the only term arising from them to the cost function $\mathcal{C}_{\mathrm{KL}}$.

However, latent variables contribute to the cost function $\mathcal{C}_{\mathrm{KL}}$ also with the part $\mathcal{C}_q$ defined in Eq. (6), resulting from the expectation $\langle \ln q(s) \rangle$. This term is

$$\mathcal{C}_{s,q} = \int_s q(s) \ln q(s) ds = -\frac{1}{2}[\ln(2\pi\widetilde{s}) + 1] \tag{11}$$

which is the negative entropy of Gaussian variable with variance $\widetilde{s}$. The parameters defining the approximation $q(s)$ of the posterior distribution of s, namely its mean $\overline{s}$ and variance $\widetilde{s}$, are to be optimised during learning.

The output of a latent Gaussian node trivially provides the mean and the variance: $\langle s \rangle = \overline{s}$ and $\operatorname{Var}\{s\} = \widetilde{s}$. The expected exponential can be easily shown to be [45, 74]

$$\langle \exp s \rangle = \exp(\overline{s} + \widetilde{s}/2) \tag{12}$$

The outputs of the nodes corresponding to the observations are known scalar values instead of distributions. Therefore for these nodes $\langle s \rangle = s$, $\operatorname{Var}\{s\} = 0$, and $\langle \exp s \rangle = \exp s$. An important conclusion of the considerations presented this far is that the cost function of a Gaussian node can be computed analytically in a closed form. This requires that the posterior approximation is Gaussian and that the mean $\langle m \rangle$ and the variance $\operatorname{Var}\{m\}$ of the mean input m as well as the mean $\langle v \rangle$ and the expected exponential $\langle \exp v \rangle$ of the variance input v can be computed. To summarise, we have shown that Gaussian nodes can be connected together and their costs can be evaluated analytically.

We will later on use derivatives of the cost function with respect to some expectations of its mean and variance parents m and v as messages from children to parents. They are derived directly from Eq. (10), taking the form

$$\frac{\partial \mathcal{C}_{s,p}}{\partial \langle m \rangle} = \langle \exp v \rangle \left(\langle m \rangle - \overline{s} \right) \tag{13}$$

$$\frac{\partial \mathcal{C}_{s,p}}{\partial \mathrm{Var}\left\{m\right\}} = \frac{\langle \exp v \rangle}{2} \tag{14}$$

$$\frac{\partial \mathcal{C}_{s,p}}{\partial \langle v \rangle} = -\frac{1}{2} \tag{15}$$

$$\frac{\partial \mathcal{C}_{s,p}}{\partial \langle \exp v \rangle} = \frac{(\overline{s} - \langle m \rangle)^2 + \mathrm{Var}\left\{m\right\} + \widetilde{s}}{2}. \tag{16}$$

4.1.2 Updating the posterior distribution

The posterior distribution $q(s)$ of a latent Gaussian node can be updated as follows.

1. The distribution $q(s)$ affects the terms of the cost function $\mathcal{C}_s$ arising from the variable s itself, namely $\mathcal{C}_{s,p}$ and $\mathcal{C}_{s,q}$, as well as the $\mathcal{C}_p$ terms of the children of s, denoted by $\mathcal{C}_{\mathrm{ch}(s),p}$. The gradients of the cost $\mathcal{C}_{\mathrm{ch}(s),p}$ with respect to $\langle s \rangle$, $\mathrm{Var}\left\{s\right\}$, and $\langle \exp s \rangle$ are computed according to Equations (13–16).

2. The terms in $\mathcal{C}_p$ which depend on $\overline{s}$ and $\widetilde{s}$ can be shown (see Appendix B.2) to be of the form [3]

$$\mathcal{C}_p = \mathcal{C}_{s,p} + \mathcal{C}_{\mathrm{ch}(s),p} = M\overline{s} + V[(\overline{s} - \overline{s}_{\mathrm{current}})^2 + \widetilde{s}] + E\left\langle \exp s \right\rangle, \tag{17}$$

   where

$$M = \frac{\partial \mathcal{C}_p}{\partial \overline{s}}, \qquad V = \frac{\partial \mathcal{C}_p}{\partial \widetilde{s}}, \quad \text{and } E = \frac{\partial \mathcal{C}_p}{\partial \langle \exp s \rangle}. \tag{18}$$

3. The minimum of $\mathcal{C}_s = \mathcal{C}_{s,p} + \mathcal{C}_{s,q} + \mathcal{C}_{\mathrm{ch}(s),p}$ is solved. This can be done analytically if $E = 0$, corresponding to the case of so-called free-form solution (see [45] for details):

$$\overline{s}_{\mathrm{opt}} = \overline{s}_{\mathrm{current}} - \frac{M}{2V}, \qquad \widetilde{s}_{\mathrm{opt}} = \frac{1}{2V}. \tag{19}$$

   Otherwise the minimum is obtained iteratively. Iterative minimisation can be carried out efficiently using Newton's method for the posterior mean $\overline{s}$ and a fixed-point iteration for the posterior variance $\widetilde{s}$. The minimisation procedure is discussed in more detail in Appendix A.

[3]Note that constants are dropped out since they do not depend on $\overline{s}$ or $\widetilde{s}$.

4.2 Addition and multiplication nodes

Consider first the addition node. The mean, variance and expected exponential of the output of the addition node can be evaluated in a straightforward way. Assuming that the inputs s_i are statistically independent, these expectations are respectively given by

$$\left\langle \sum_{i=1}^{n} s_i \right\rangle = \sum_{i=1}^{n} \langle s_i \rangle \tag{20}$$

$$\operatorname{Var}\left\{ \sum_{i=1}^{n} s_i \right\} = \sum_{i=1}^{n} \operatorname{Var}\{s_i\} \tag{21}$$

$$\left\langle \exp\left(\sum_{i=1}^{n} s_i \right) \right\rangle = \prod_{i=1}^{n} \langle \exp s_i \rangle \tag{22}$$

The proof has been given in Appendix B.1.

Consider then the multiplication node. Assuming independence between the inputs s_i, the mean and the variance of the output take the form (see Appendix B.1)

$$\left\langle \prod_{i=1}^{n} s_i \right\rangle = \prod_{i=1}^{n} \langle s_i \rangle \tag{23}$$

$$\operatorname{Var}\left\{ \prod_{i=1}^{n} s_i \right\} = \prod_{i=1}^{n} \left[\langle s_i \rangle^2 + \operatorname{Var}\{s_i\} \right] - \prod_{i=1}^{n} \langle s_i \rangle^2 \tag{24}$$

For the multiplication node the expected exponential cannot be evaluated without knowing the exact distribution of the inputs.

The formulas (20)–(24) are given for n inputs because of generality, but in practice we have carried out the needed calculations pairwise. When using the general formula (24), the variance might otherwise occasionally take a small negative value due to minor imprecisions appearing in the computations. This problem does not arise in pairwise computations. Now, the propagation in the forward direction is covered.

The form of the cost function propagating from children to parents is assumed to be of the form (17). This is true even in the case, where there are addition and multiplication nodes in between (see Appendix B.2 for proof). Therefore only the gradients of the cost function with respect to the different expectations need to be propagated backwards to identify the whole cost function w.r.t. the parent. The required formulas are obtained in a straightforward manner from Eqs. (20)–(24). The gradients for the addition node are:

$$\frac{\partial C}{\partial \langle s_1 \rangle} = \frac{\partial C}{\partial \langle s_1 + s_2 \rangle} \tag{25}$$

$$\frac{\partial C}{\partial \mathrm{Var}\left\{s_1\right\}} = \frac{\partial C}{\partial \mathrm{Var}\left\{s_1 + s_2\right\}} \tag{26}$$

$$\frac{\partial C}{\partial \left\langle \exp s_1 \right\rangle} = \left\langle \exp s_2 \right\rangle \frac{\partial C}{\partial \left\langle \exp(s_1 + s_2) \right\rangle}. \tag{27}$$

For the multiplication node, they become

$$\frac{\partial C}{\partial \left\langle s_1 \right\rangle} = \left\langle s_2 \right\rangle \frac{\partial C}{\partial \left\langle s_1 s_2 \right\rangle} + 2\mathrm{Var}\left\{s_2\right\} \frac{\partial C}{\partial \mathrm{Var}\left\{s_1 s_2\right\}} \left\langle s_1 \right\rangle \tag{28}$$

$$\frac{\partial C}{\partial \mathrm{Var}\left\{s_1\right\}} = \left(\left\langle s_2 \right\rangle^2 + \mathrm{Var}\left\{s_2\right\} \right) \frac{\partial C}{\partial \mathrm{Var}\left\{s_1 s_2\right\}}. \tag{29}$$

As a conclusion, addition and multiplication nodes can be added between the Gaussian nodes whose costs still retain the form (17). Proofs can be found in Appendices B.1 and B.2.

4.3 Nonlinearity node

A serious problem arising here is that for most nonlinear functions it is impossible to compute the required expectations analytically. Here we describe a particular nonlinearity in detail and discuss the options for extending to other nonlinearities, for which the implementation is underway.

Ghahramani and Roweis have shown [19] that for the nonlinear function $f(s) = \exp(-s^2)$ in Eq. (9), the mean and variance have analytical expressions, to be presented shortly, provided that it has Gaussian input. In our graphical network structures this condition is fulfilled if we require that the nonlinearity must be inserted immediately after a Gaussian node. The same type of exponential function (9) is frequently used in standard radial-basis function networks [8, 24, 19], but in a different manner. There the exponential function depends on the Euclidean distance from a center point, while in our case it depends on the input variable s directly.

The first and second moments of the function (9) with respect to the distribution $q(s)$ are [19]

$$\left\langle f(s) \right\rangle = \exp\left(-\frac{\overline{s}^2}{2\widetilde{s} + 1} \right) (2\widetilde{s} + 1)^{-\frac{1}{2}} \tag{30}$$

$$\left\langle [f(s)]^2 \right\rangle = \exp\left(-\frac{2\overline{s}^2}{4\widetilde{s} + 1} \right) (4\widetilde{s} + 1)^{-\frac{1}{2}} \tag{31}$$

The formula (30) provides directly the mean $\left\langle f(s) \right\rangle$, and the variance is obtained from (30) and (31) by applying the familiar formula $\mathrm{Var}\left\{f(s)\right\} = \left\langle [f(s)]^2 \right\rangle - \left\langle f(s) \right\rangle^2$. The expected exponential $\left\langle \exp f(s) \right\rangle$ cannot be evaluated analytically, which limits somewhat the use of the nonlinear node.

The updating of the nonlinear node following directly a Gaussian node takes place similarly as the updating of a plain Gaussian node. The gradients of $\mathcal{C}_p$ with respect to $\left\langle f(s) \right\rangle$ and $\mathrm{Var}\left\{f(s)\right\}$ are evaluated assuming that they arise from a quadratic term. This assumption holds since the nonlinearity can only propagate to the mean of Gaussian nodes. The update formulas are given in Appendix C.

Another possibility is to use as the nonlinearity the error function $f(s) = \int_{-\infty}^{s} \exp(-r^2)dr$, because its mean can be evaluated analytically and variance approximated from above [15]. Increasing the variance increases the value of the cost function, too, and hence it suffices to minimise the upper bound of the cost function for finding a good solution. [15] apply the error function in MLP (multilayer perceptron) networks [8, 24] but in a manner different from ours.

Finally, [54] has applied the hyperbolic tangent function $f(s) = \tanh(s)$, approximating it iteratively with a Gaussian. [32] approximate the same sigmoidal function with a Gauss-Hermite quadrature. This alternative could be considered here, too. A problem with it is, however, that the cost function (mean and variance) cannot be computed analytically.

4.4 Other possible nodes

One of the authors has recently implemented two new variable nodes [21, 23] into the Bayes Blocks software library. They are the mixture-of-Gaussians (MoG) node and the rectified Gaussian node. MoG can be used to model any sufficiently well behaving distribution [8]. In the independent factor analysis (IFA) method introduced in [2], a MoG distribution was used for the sources, resulting in a probabilistic version of independent component analysis (ICA) [36].

The second new node type, the rectified Gaussian variable, was introduced in [53]. By omitting negative values and retaining only positive ones of a variable which is originally Gaussian distributed, this block allows modelling of variables having positive values only. Such variables are commonplace for example in digital image processing, where the picture elements (pixels) have always non-negative values. The cost functions and update rules of the MoG and rectified Gaussian node have been derived in [21]. We postpone a more detailed discussion of these nodes to forthcoming papers to keep the length of this paper reasonable.

In the early conference paper [77] where we introduced the blocks for the first time, two more blocks were proposed for handling discrete models and variables. One of them is a switch, which picks up its k-th continuous valued input signal as its output signal. The other one is a discrete variable k, which has a soft-max prior derived from the continuous valued input signals c_i of the node. However, we have omitted these two nodes from the present paper, because their performance has not turned out to be adequate. The reason might be that assuming all parents of all nodes independent is too restrictive. For instance, building a mixture-of-Gaussians from discrete and Gaussian variables with switches is possible, but the construction loses

Node type	$\langle\cdot\rangle$	$\mathrm{Var}\{\cdot\}$	$\langle\exp\cdot\rangle$
Gaussian node	$\overline{s}$	$\widetilde{s}$	(12)
Addition node	(20)	(21)	(22)
Multiplication node	(23)	(24)	-
Nonlinearity	(30)	(30),(31)	-
Constant	c	0	$\exp c$

Table 1: The forward messages or expectations that are provided by the output of different types of nodes. The numbers in parentheses refer to defining equations. The multiplication and nonlinearity cannot provide the expected exponential.

Input type	$\frac{\partial\mathcal{C}}{\partial\langle\cdot\rangle}$	$\frac{\partial\mathcal{C}}{\partial\mathrm{Var}\{\cdot\}}$	$\frac{\partial\mathcal{C}}{\partial\langle\exp\cdot\rangle}$
Mean of a Gaussian node	(13)	(14)	0
Variance of a Gaussian node	(15)	0	(16)
Addendum	(25)	(26)	(27)
Factor	(28)	(29)	0

Table 2: The backward messages or the gradients of the cost function w.r.t. certain expectations. The numbers in parentheses refer to defining equations. The gradients of the Gaussian node are derived from Eq. (10). The Gaussian node requires the corresponding expectations from its inputs, that is, $\langle m\rangle$, $\mathrm{Var}\{m\}$, $\langle v\rangle$, and $\langle\exp v\rangle$. Addition and multiplication nodes require the same type of input expectations that they are required to provide as output. Communication of a nonlinearity with its Gaussian parent node is described in Appendix C.

out to a specialised MoG node that makes fewer assumptions. In [63], the discrete node is used without switches.

Action and utility nodes [60, 55] would extend the library into decision theory and control. In addition to the messages about the variational Bayesian cost function, the network would propagate messages about utility. [64] describe such a system in a slightly different framework.

5 Combining the nodes

The expectations provided by the outputs and required by the inputs of the different nodes are summarised in Tables 1 and 2, respectively. One can see that the variance input of a Gaussian node requires the expected exponential of the incoming signal. However, it cannot be computed for the nonlinear and multiplication nodes. Hence all the nodes cannot be combined freely.

When connecting the nodes, the following restrictions must be taken into account:

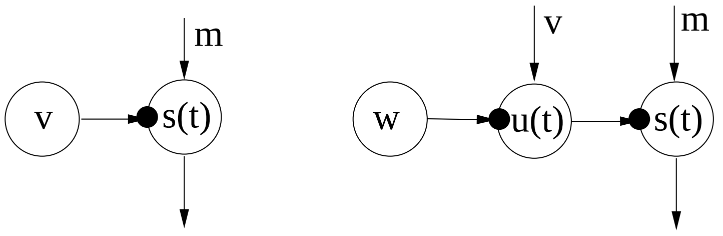


Figure 3: *Left:* The Gaussian variable $s(t)$ has a a constant variance $\exp(-v)$ and mean m. *Right:* A variance source is added for providing a non-constant variance input $u(t)$ to the output (source) signal $s(t)$. The variance source $u(t)$ has a prior mean v and prior variance $\exp(-w)$.

1. In general, the network has to be a directed acyclic graph (DAG). The delay nodes are an exception because the past values of any node can be the parents of any other nodes. This violation is not a real one in the sense that if the structure were unfolded in time, the resulting network would again be a DAG.

2. The nonlinearity must always be placed immediately after a Gaussian node. This is because the output expectations, Equations (30) and (31), can be computed only for Gaussian inputs. The nonlinearity also breaks the general form of the likelihood (17). This is handled by using special update rules for the Gaussian followed by a nonlinearity (Appendix C).

3. The outputs of multiplication and nonlinear nodes cannot be used as variance inputs for the Gaussian node. This is because the expected exponential cannot be evaluated for them. These restrictions are evident from Tables 1 and 2.

4. There should be only one computational path from a latent variable to a variable. Otherwise, the independency assumptions used in Equations (10) and (21)–(24) are violated and variational Bayesian learning becomes more complicated (recall Figure 1).

Note that the network may contain loops, that is, the underlying undirected network can be cyclic. Note also that the second, third, and fourth restrictions can be circumvented by inserting mediating Gaussian nodes. A mediating Gaussian node that is used as the variance input of another variable, is called the variance source and it is discussed in the following.

5.1 Nonstationary variance

In most currently used models, only the means of Gaussian nodes have hierarchical or dynamical models. In many real-world situations the variance

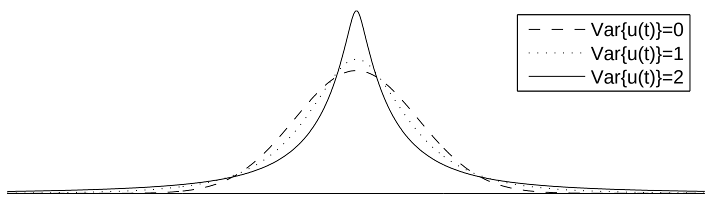


Figure 4: The distribution of $s(t)$ is plotted when $s(t) \sim \mathcal{N}(0, \exp[-u(t)])$ and $u(t) \sim \mathcal{N}(0, \cdot)$. Note that when $\mathrm{Var}\{u(t)\} = 0$, the distribution of $s(t)$ is Gaussian. This corresponds to the right subfigure of Fig. 3 when $m = v = 0$ and $\exp(-w) = 0, 1, 2$.

is not a constant either but it is more difficult to model it. For modelling the variance, too, we use the variance source [71] depicted schematically in Figure 3. Variance source is a regular Gaussian node whose output $u(t)$ is used as the input variance of another Gaussian node. Variance source can convert prediction of the mean into prediction of the variance, allowing to build hierarchical or dynamical models for the variance.

The output $s(t)$ of a Gaussian node to which the variance source is attached (see the right subfigure of Fig. 3) has in general a super-Gaussian distribution. Such a distribution is typically characterised by long tails and a high peak, and it is formally defined as having a positive value of kurtosis (see [36] for a detailed discussion). This property has been proved for example in [59], where it is shown that a nonstationary variance (amplitude) always increases the kurtosis. The output signal $s(t)$ of the stationary Gaussian variance source depicted in the left subfigure of Fig. 3 is naturally Gaussian distributed with zero kurtosis. The variance source is useful in modelling natural signals such as speech and images which are typically super-Gaussian, and also in modelling outliers in the observations.

5.2 Linear independent factor analysis

In many instances there exist several nodes which have quite similar role in the chosen structure. Assuming that i^{th} such node corresponds to a scalar variable y_i, it is convenient to use the vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$ to jointly denote all the corresponding scalar variables $y_1, y_2, \ldots, y_n$. This notation is used in Figures 5 and 6 later on. Hence we represent the scalar source nodes corresponding to the variables $s_i(t)$ using the source vector $\mathbf{s}(t)$, and the scalar nodes corresponding to the observations $x_i(t)$ using the observation vector $\mathbf{x}(t)$.

The addition and multiplication nodes can be used for building an affine

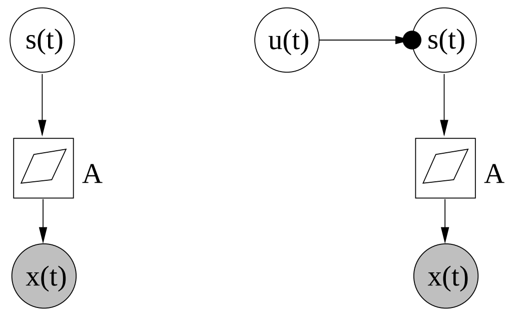


Figure 5: Model structures for linear factor analysis (FA) (left) and independent factor analysis (IFA) (right).

transformation

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}_x(t) \tag{32}$$

from the Gaussian source nodes $\mathbf{s}(t)$ to the Gaussian observation nodes $\mathbf{x}(t)$. The vector $\mathbf{a}$ denotes the bias and vector $\mathbf{n}_x(t)$ denotes the zero-mean Gaussian noise in the Gaussian node $\mathbf{x}(t)$. This model corresponds to standard linear factor analysis (FA) assuming that the sources $s_i(t)$ are mutually uncorrelated; see for example [36].

If instead of Gaussianity it is assumed that each source $s_i(t)$ has some non-Gaussian prior, the model (32) describes linear independent factor analysis (IFA). Linear IFA was introduced by [2], who used variational Bayesian learning for estimating the model except for some parts which he estimated using the expectation-maximisation (EM) algorithm. Attias used a mixture-of-Gaussians source model, but another option is to use the variance source to achieve a super-Gaussian source model. Figure 5 depicts the model structures for linear factor analysis and independent factor analysis.

5.3 A hierarchical variance model

Figure 6 (right subfigure) presents a hierarchical model for the variance, and also shows how it can be constructed by first learning simpler structures shown in the left and middle subfigures of Fig. 6. This is necessary, because learning a hierarchical model having different types of nodes from scratch in a completely unsupervised manner would be too demanding a task, ending quite probably into an unsatisfactory local minimum.

The final rightmost variance model in Fig. 6 is somewhat involved in that it contains both nonlinearities and hierarchical modelling of variances. Before going into its mathematical details and into the two simpler models in Fig. 6, we point out that we have considered in our earlier papers related but simpler block models. In [76], a hierarchical nonlinear model for the data $\mathbf{x}(t)$ is discussed without modelling the variance. Such a model can be

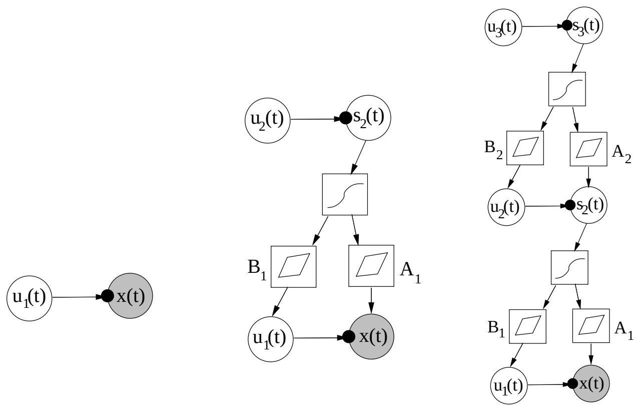


Figure 6: Construction of a hierarchical variance model in stages from simpler models. *Left:* In the beginning, a variance source is attached to each Gaussian observation node. The nodes represent vectors. *Middle:* A layer of sources with variance sources attached to them is added. They layers are connected through a nonlinearity and an affine mapping. *Right:* Another layer is added on the top to form the final hierarchical variance model.

applied for example to nonlinear ICA or blind source separation. Experimental results [76] show that this block model performs adequately in the nonlinear BSS problem, even though the results are slightly poorer than for our earlier computationally more demanding model [44, 75, 32] with multiple computational paths.

In another paper [71], we have considered hierarchical modelling of variance using the block approach without nonlinearities. Experimental results on biomedical MEG (magnetoencephalography) data demonstrate the usefulness of hierarchical modelling of variances and existence of variance sources in real-world data.

Learning starts from the simple structure shown in the left subfigure of Fig. 6. There a variance source is attached to each Gaussian observation node. The nodes represent vectors, with $\mathbf{u}_1(t)$ being the output vector of the variance source and $\mathbf{x}(t)$ the t^{th} observation (data) vector. The vectors $\mathbf{u}_1(t)$ and $\mathbf{x}(t)$ have the same dimension, and each component of the variance vector $\mathbf{u}_1(t)$ models the variance of the respective component of the observation vector $\mathbf{x}(t)$.

Mathematically, this simple first model obeys the equations

$$\mathbf{x}(t) = \mathbf{a}_1 + \mathbf{n}_x(t) \tag{33}$$

$$\mathbf{u}_1(t) = \mathbf{b}_1 + \mathbf{n}_{u_1}(t) \tag{34}$$

Here the vectors $\mathbf{a}_1$ and $\mathbf{b}_1$ denote the constant means (bias terms) of the data vector $\mathbf{x}(t)$ and the variance variable vector $\mathbf{u}_1(t)$, respectively. The additive "noise" vector $\mathbf{n}_x(t)$ determines the variances of the components of $\mathbf{x}(t)$. It has a Gaussian distribution with a zero mean and variance $\exp[-\mathbf{u}_1(t)]$:

$$\mathbf{n}_x(t) \sim \mathcal{N}(\mathbf{0}, \exp[-\mathbf{u}_1(t)]) \tag{35}$$

More precisely, the shorthand notation $\mathcal{N}(\mathbf{0}, \exp[-\mathbf{u}_1(t)])$ means that each component of $\mathbf{n}_x(t)$ is Gaussian distributed with a zero mean and variance defined by the respective component of the vector $\exp[-\mathbf{u}_1(t)]$. The exponential function $\exp(\cdot)$ is applied separately to each component of the vector $-\mathbf{u}_1(t)$. Similarly,

$$\mathbf{n}_{u_1}(t) \sim \mathcal{N}(\mathbf{0}, \exp[-\mathbf{v}_1]) \tag{36}$$

where the components of the vector $\mathbf{v}_1$ define the variances of the zero mean Gaussian variables $\mathbf{n}_{u_1}(t)$.

Consider then the intermediate model shown in the middle subfigure of Fig. 6. In this second learning stage, a layer of sources with variance sources attached to them is added. These sources are represented by the source vector $\mathbf{s}_2(t)$, and their variances are given by the respective components of the variance vector $\mathbf{u}_2(t)$ quite similarly as in the left subfigure. The (vector) node between the source vector $\mathbf{s}_2(t)$ and the variance vector $\mathbf{u}_1(t)$ represents an affine transformation with a transformation matrix $\mathbf{A}_1$ including a bias term. Hence the prior mean inputted to the Gaussian variance source having the output $\mathbf{u}_1(t)$ is of the form $\mathbf{B}_1\mathbf{f}(\mathbf{s}_2(t)) + \mathbf{b}_1$, where $\mathbf{b}_1$ is the bias vector, and $\mathbf{f}(\cdot)$ is a vector of componentwise nonlinear functions (9). Quite similarly, the vector node between $\mathbf{s}_2(t)$ and the observation vector $\mathbf{x}(t)$ yields as its output the affine transformation $\mathbf{A}_1\mathbf{f}(\mathbf{s}_2(t)) + \mathbf{a}_1$, where $\mathbf{a}_1$ is a bias vector. This in turn provides the input prior mean to the Gaussian node modelling the observation vector $\mathbf{x}(t)$.

The mathematical equations corresponding to the model represented graphically in the middle subfigure of Fig. 6 are:

$$\mathbf{x}(t) = \mathbf{A}_1\mathbf{f}(\mathbf{s}_2(t)) + \mathbf{a}_1 + \mathbf{n}_x(t) \tag{37}$$

$$\mathbf{u}_1(t) = \mathbf{B}_1\mathbf{f}(\mathbf{s}_2(t)) + \mathbf{b}_1 + \mathbf{n}_{u_1}(t) \tag{38}$$

$$\mathbf{s}_2(t) = \mathbf{a}_2 + \mathbf{n}_{s_2}(t) \tag{39}$$

$$\mathbf{u}_2(t) = \mathbf{b}_2 + \mathbf{n}_{u_2}(t) \tag{40}$$

Compared with the simplest model (33)–(34), one can observe that the source vector $\mathbf{s}_2(t)$ of the second (upper) layer and the associated variance vector $\mathbf{u}_2(t)$ are of quite similar form, given in Eqs. (39)–(40). The models (37)–(38) of the data vector $\mathbf{x}(t)$ and the associated variance vector $\mathbf{u}_1(t)$ in the first (bottom) layer differ from the simple first model (33)–(34) in that they contain additional terms $\mathbf{A}_1\mathbf{f}(\mathbf{s}_2(t))$ and $\mathbf{B}_1\mathbf{f}(\mathbf{s}_2(t))$, respectively. In these terms, the nonlinear transformation $\mathbf{f}(\mathbf{s}_2(t))$ of the source vector $\mathbf{s}_2(t)$ coming from the upper layer have been multiplied by the linear mixing matrices $\mathbf{A}_1$ and $\mathbf{B}_1$. All the "noise" terms $\mathbf{n}_x(t)$, $\mathbf{n}_{u_1}(t)$, $\mathbf{n}_{s_2}(t)$, and $\mathbf{n}_{u_2}(t)$

in Eqs. (37)–(40) are modelled by similar zero mean Gaussian distributions as in Eqs. (35) and (36).

In the last stage of learning, another layer is added on the top of the network shown in the middle subfigure of Fig. 6. The resulting structure is shown in the right subfigure. The added new layer is quite similar as the layer added in the second stage. The prior variances represented by the vector $\mathbf{u}_3(t)$ model the source vector $\mathbf{s}_3(t)$, which is turn affects via the affine transformation $\mathbf{B}_2\mathbf{f}(\mathbf{s}_3(t)) + \mathbf{b}_2$ to the mean of the mediating variance node $\mathbf{u}_2(t)$. The source vector $\mathbf{s}_3(t)$ provides also the prior mean of the source $\mathbf{s}_2(t)$ via the affine transformation $\mathbf{A}_2\mathbf{f}(\mathbf{s}_3(t)) + \mathbf{a}_2$.

The model equations (37)–(38) for the data vector $\mathbf{x}(t)$ and its associated variance vector $\mathbf{u}_1(t)$ remain the same as in the intermediate model shown graphically in the middle subfigure of Fig. 6. The model equations of the second and third layer sources $\mathbf{s}_2(t)$ and $\mathbf{s}_3(t)$ as well as their respective variance vectors $\mathbf{u}_2(t)$ and $\mathbf{u}_3(t)$ in the rightmost subfigure of Fig. 6 are given by

$$\mathbf{s}_2(t) = \mathbf{A}_2\mathbf{f}(\mathbf{s}_3(t)) + \mathbf{a}_2 + \mathbf{n}_{s_2}(t) \tag{41}$$

$$\mathbf{u}_2(t) = \mathbf{B}_2\mathbf{f}(\mathbf{s}_3(t)) + \mathbf{b}_2 + \mathbf{n}_{u_2}(t) \tag{42}$$

$$\mathbf{s}_3(t) = \mathbf{a}_3 + \mathbf{n}_{s_3}(t) \tag{43}$$

$$\mathbf{u}_3(t) = \mathbf{b}_3 + \mathbf{n}_{u_3}(t) \tag{44}$$

Again, the vectors $\mathbf{a}_2$, $\mathbf{b}_2$, $\mathbf{a}_3$, and $\mathbf{b}_3$ represent the constant means (biases) in their respective models, and $\mathbf{A}_2$ and $\mathbf{B}_2$ are mixing matrices with matching dimensions. The vectors $\mathbf{n}_{s_2}(t)$, $\mathbf{n}_{u_2}(t)$, $\mathbf{n}_{s_3}(t)$, and $\mathbf{n}_{u_3}(t)$ have similar zero mean Gaussian distributions as in Eqs. (35) and (36).

It should be noted that in the resulting network the number of scalar-valued nodes (size of the layers) can be different for different layers. Additional layers could be appended in the same manner. The final network of the right subfigure in Fig. 6 utilises variance nodes in building a hierarchical model for both the means and variances. Without the variance sources the model would correspond to a nonlinear model with latent variables in the hidden layer. As already mentioned, we have considered such a nonlinear hierarchical model in [76]. Note that computation nodes as hidden nodes would result in multiple paths from the latent variables of the upper layer to the observations. This type of structure was used in [44], and it has a quadratic computational complexity as opposed to linear one of the networks in Figure 6.

5.4 Linear dynamic models for the sources and variances

Sometimes it is useful to complement the linear factor analysis model

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}_x(t) \tag{45}$$

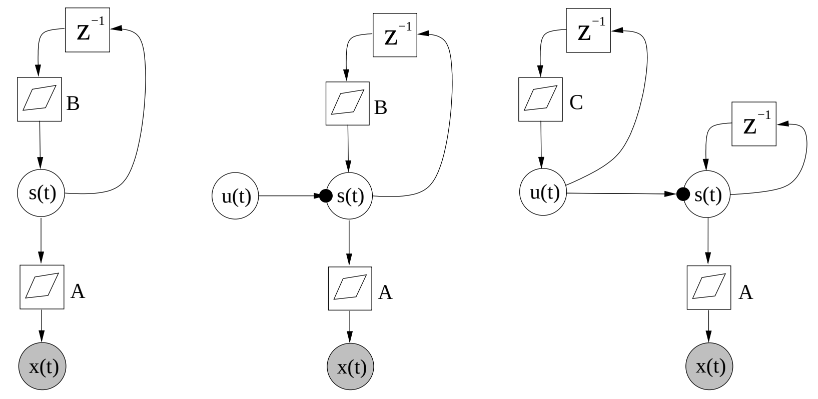


Figure 7: Three model structures. A linear Gaussian state-space model (left); the same model complemented with a super-Gaussian innovation process for the sources (middle); and a dynamic model for the variances of the sources which also have a recurrent dynamic model (right).

with a recursive one-step prediction model for the source vector $\mathbf{s}(t)$:

$$\mathbf{s}(t) = \mathbf{B}\mathbf{s}(t-1) + \mathbf{b} + \mathbf{n}_s(t) \tag{46}$$

The noise term $\mathbf{n}_s(t)$ is called the innovation process. The dynamic model of the type (45), (46) is used for example in Kalman filtering [24, 25], but other estimation algorithms can be applied as well [24]. The left subfigure in Fig. 7 depicts the structure arising from Eqs. (45) and (46), built from the blocks.

A straightforward extension is to use variance sources for the sources to make the innovation process super-Gaussian. The variance signal $\mathbf{u}(t)$ characterises the innovation process of $\mathbf{s}(t)$, in effect telling how much the signal differs from the predicted one but not in which direction it is changing. The graphical model of this extension is depicted in the middle subfigure of Fig. 7. The mathematical equations describing this model can be written in a similar manner as for the hierarchical variance models in the previous subsection.

Another extension is to model the variance sources dynamically by using one-step recursive prediction model for them:

$$\mathbf{u}(t) = \mathbf{C}\mathbf{u}(t-1) + \mathbf{c} + \mathbf{n}_u(t). \tag{47}$$

This model is depicted graphically in the rightmost subfigure of Fig. 7. In context with it, we use the simplest possible identity dynamical mapping for $\mathbf{s}(t)$:

$$\mathbf{s}(t) = \mathbf{s}(t-1) + \mathbf{n}_s(t). \tag{48}$$

The latter two models introduced in this subsection will be tested experimentally later on in this paper.

5.5 Hierarchical priors

It is often desirable that the priors of the parameters should not be too restrictive. A common type of a vague prior is the hierarchical prior [16]. For example the priors of the elements a_{ij} of a mixing matrix $\mathbf{A}$ can be defined via the Gaussian distributions

$$p(a_{ij} \mid v_i^a) = \mathcal{N}(a_{ij}; 0, \exp(-v_i^a)) \tag{49}$$

$$p(v_i^a \mid m^{va}, v^{va}) = \mathcal{N}(v_i^a; m^{va}, \exp(-v^{va})) \,. \tag{50}$$

Finally, the priors of the quantities m^{va} and v^{va} have flat Gaussian distributions $\mathcal{N}(\cdot; 0, 100)$ (the constants depending on the scale of the data). When going up in the hierarchy, we use the same distribution for each column of a matrix and for each component of a vector. On the top, the number of required constant priors is small. Thus very little information is provided and needed a priori. This kind of hierarchical priors are used in the experiments later on this paper.

6 Learning

Let us now discuss the overall learning procedure, describing also briefly how problems related with learning can be handled.

6.1 Updating of the network

The nodes of the network communicate with their parents and children by providing certain expectations in the feedforward direction (from parents to children) and gradients of the cost function with respect to the same expectations in the feedback direction (from children to parents). These expectations and gradients are summarised in Tables 1 and 2.

The basic element for updating the network is the update of a single node assuming the rest of the network fixed. For computation nodes this is simple: each time when a child node asks for expectations and they are out of date, the computational node asks from its parents for their expectations and updates its own ones. And vice versa: when parents ask for gradients and they are out of date, the node asks from its children for the gradients and updates its own ones. These updates have analytical formulas given in Section 4.

For a variable node to be updated, the input expectations and output gradients need to be up-to-date. The posterior approximation $q(s)$ can then be adjusted to minimise the cost function as explained in Section 4. The minimisation is either analytical or iterative, depending on the situation. Signals propagating outwards from the node (the output expectations and the input gradients) of a variable node are functions of $q(s)$ and are thus updated in the process. Each update is guaranteed not to increase the cost function.

One sweep of updating means updating each node once. The order in which this is done is not critical for the system to work. It would not be useful to update a variable twice without updating some of its neighbours in between, but that does not happen with any ordering when updates are done in sweeps. We have used an ordering where each variable node is updated only after all of its descendants have been updated. Basically when a variable node is updated, its input gradients and output expectations are labeled as outdated and they are updated only when another node asks for that information.

It is possible to use different measures to improve the learning process. Measures for avoiding local minima are described in the next subsection. Another enhancement can be used for speeding up learning. The basic idea is that after some time, the parameters describing $q(s)$ are changing fairly linearly between consecutive sweeps. Therefore a line search in that direction provides faster learning, as discussed in [28, 33]. We apply this line search only at every tenth sweep for allowing the consecutive updates to become fairly linear again.

Learning a model typically takes thousands of sweeps before convergence. The cost function decreases monotonically after every update. Typically this decrease gets smaller with time, but not always monotonically. Therefore care should be taken in selecting the stopping criterion. We have chosen to stop the learning process when the decrease in the cost during the previous 200 sweeps is lower than some predefined threshold.

6.2 Structural learning and local minima

The chosen model has a pre-specified structure which, however, has some flexibility. The number of nodes is not fixed in advance, but their optimal number is estimated using variational Bayesian learning, and unnecessary connections can be pruned away.

A factorial posterior approximation, which is used in this paper, often leads to automatic pruning of some of the connections in the model. When there is not enough data to estimate all the parameters, some directions remain ill-determined. This causes the posterior distribution along those directions to be roughly equal to the prior distribution. In variational Bayesian learning with a factorial posterior approximation, the ill-determined directions tend to get aligned with the axes of the parameter space because then the factorial approximation is most accurate.

The pruning tendency makes it easy to use for instance sparsely connected models, because the learning algorithm automatically selects a small amount of well-determined parameters. But at the early stages of learning, pruning can be harmful, because large parts of the model can get pruned away before a sensible representation has been found. This corresponds to the situation where the learning scheme ends up into a local minimum of the cost function [50]. A posterior approximation which takes into account the posterior dependences has the advantage that it has far less local minima than a factorial posterior approximation. It seems that Bayesian learning algorithms which have linear time complexity cannot avoid local minima in general.

However, suitable choices of the model structure and countermeasures included in the learning scheme can alleviate the problem greatly. We have used the following means for avoiding getting stuck into local minima:

- Learning takes place in several stages, starting from simpler structures which are learned first before proceeding to more complicated hierarchic structures. An example of this technique was presented in Section 5.3.

- New parts of the network are initialised appropriately. One can use for instance principal component analysis (PCA), independent component analysis (ICA), vector quantisation, or kernel PCA [29]. The best option depends on the application. Often it is useful to try different methods and select the one providing the smallest value of the cost function for the learned model. There are two ways to handle initialisation: either to fix the sources for a while and learn the weights of the model, or to fix the weights for a while and learn the sources corresponding to the observations. The fixed variables can be released gradually (see Section 5.1 of [76]).

- Automatic pruning is discouraged initially by omitting the term

  $$2\text{Var}\left\{s_2\right\}\frac{\partial C}{\partial \text{Var}\left\{s_1 s_2\right\}}\langle s_1\rangle$$

  in the multiplication nodes (Eq. (28)). This effectively means that the mean of s_1 is optimistically adjusted as if there were no uncertainty about s_2. In this way the cost function may increase at first due to overoptimism, but it may pay off later on by escaping early pruning.

- New sources $s_i(t)$ (components of the source vector $\mathbf{s}(t)$ of a layer) are generated, and pruned sources are removed from time to time.

- The activations of the sources are reset a few times. The sources are re-adjusted to their places while keeping the mapping and other parameters fixed. This often helps if some of the sources are stuck into a local minimum.

7 Experimental results

The Bayes Blocks software [72] has been applied to several problems.

[71] considered several models of variance. The main application was the analysis of MEG measurements from a human brain. In addition to features corresponding to brain activity the data contained several artifacts such as muscle activity induced by the patient biting his teeth. Linear ICA applied to the data was able to separate the original causes to some degree but still many dependencies remained between the sources. Hence an additional layer of so-called variance sources was used to find correlations between the variances of the innovation processes of the ordinary sources. These were able to capture phenomena related to the biting artifact as well as to rhythmic activity.

An astrophysical problem of separating young and old star populations from a set of elliptical galaxy spectra has been studied by one of the authors in [57]. Since the observed quantities are energies and thus positive and since the mixing process is also known to be positive, it is necessary for the subsequent astrophysical analysis to be feasible to include these constraints to the model as well. The standard technique of putting a positive prior on the sources was found to have the unfortunate technical shortcoming of inducing sparsely distributed factors, which was deemed inappropriate in that specific application. To get rid of the induced sparsity but to still keep the positivity constraint, the nonnegativity was forced by rectification nonlinearities [22]. In addition to finding an astrophysically meaningful factorisation, several other specifications were needed to be met related to handling of missing values, measurements errors and predictive capabilities of the model.

In [63], a nonlinear model for relational data is applied to the analysis of the boardgame Go. The difficult part of the game state evaluation is to determine which groups of stones are likely to get captured. A model similar to the one that will be described in Section 7.2, is built for features of pairs of groups, including the probability of getting captured. When the learned model is applied to new game states, the estimates propagate through a network of such pairs. The structure of the network is thus determined by the game state. The approach can be used for inference in relational databases.

The following three sets of experiments are given as additional examples. The first one is a difficult toy problem that illustrates hierarchy and variance modelling, the second one studies the inference of missing values in speech spectra, and the third one has a dynamical model for image sequences.

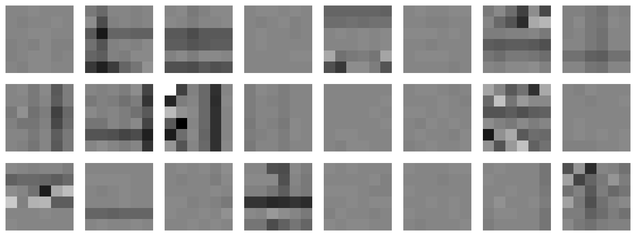


Figure 8: Samples from the 1000 image patches used in the extended bars problem. The bars include both standard and variance bars in horizontal and vertical directions. For instance, the patch at the bottom left corner shows the activation of a standard horizontal bar above the horizontal variance bar in the middle.

7.1 Bars problem

The first experimental problem studied was testing of the hierarchical nonlinear variance model in Figure 6 in an extension of the well-known bars problem [13]. The data set consisted of 1000 image patches each having 6×6 pixels. They contained both horizontal and vertical bars. In addition to the regular bars, the problem was extended to include horizontal and vertical variance bars, characterized and manifested by their higher variance. Samples of the image patches used are shown in Figure 8.

The data were generated by first choosing whether vertical, horizontal, both, or neither orientations were active, each with probability 1/4. Whenever an orientation is active, there is a probability 1/3 for a bar in each row or column to be active. For both orientations, there are 6 regular bars, one for each row or column, and 3 variance bars which are 2 rows or columns wide. The intensities (grey level values) of the bars were drawn from a normalised positive exponential distribution having the pdf $p(z) = \exp(-z), z \geq 0$, $p(z) = 0, z < 0$. Regular bars are additive, and variance bars produce additive Gaussian noise having the standard deviation of its intensity. Finally, Gaussian noise with a standard deviation 0.1 was added to each pixel.

The network was built up following the stages shown in Figure 6. It was initialised with a single layer with 36 nodes corresponding to the 36 dimensional data vector. The second layer of 30 nodes was created at the sweep 20, and the third layer of 5 nodes at the sweep 100. After creating a layer only its sources were updated for 10 sweeps, and pruning was discouraged for 50 sweeps. New nodes were added twice, 3 to the second layer and 2 to the third layer, at sweeps 300 and 400. After that, only the sources were updated for 5 sweeps, and pruning was again discouraged for 50 sweeps. The source activations were reset at the sweeps 500, 600 and 700, and only the sources were updated for the next 40 sweeps. Dead nodes were removed every 20 sweeps. The multistage training procedure was designed to avoid suboptimal local solutions, as discussed in Section 6.2.

Figure 9 demonstrates that the algorithm finds a generative model that is quite similar to the generation process. The two sources on the third layer correspond to the horizontal and vertical orientations and the 18 sources on the second layer correspond to the bars. Each element of the weight matrices is depicted as a pixel with the appropriate grey level value in Fig. 9. The pixels of $\mathbf{A}_2$ and $\mathbf{B}_2$ are ordered similarly as the patches of $\mathbf{A}_1$ and $\mathbf{B}_1$, that is, vertical bars on the left and horizontal bars on the right. Regular bars, present in the mixing matrix $\mathbf{A}_1$, are reconstructed accurately, but the variance bars in the mixing matrix $\mathbf{B}_1$ exhibit some noise. The distinction between horizontal and vertical orientations is clearly visible in the mixing matrix $\mathbf{A}_2$.

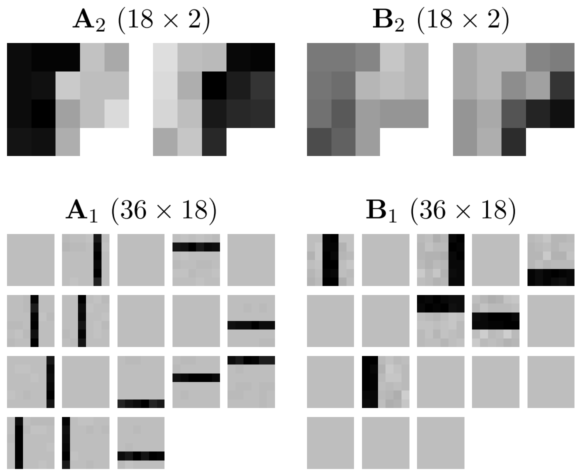


Figure 9: Results of the extended bars problem: Posterior means of the weight matrices after learning. The sources of the second layer have been ordered for visualisation purposes according to the weight (mixing) matrices $\mathbf{A}_2$ and $\mathbf{B}_2$. The elements of the matrices have been depicted as pixels having corresponding grey level values. The 18 pixels in the weight matrices $\mathbf{A}_2$ and $\mathbf{B}_2$ correspond to the 18 patches in the weight matrices $\mathbf{A}_1$ and $\mathbf{B}_1$.

A comparison experiment with a simplified learning procedure was run to demonstrate the importance of local optima. The creation and pruning of layers were done as before, but other methods for avoiding local minima (addition of nodes, discouraging pruning and resetting of sources) were disabled. The resulting weights can be seen in Figure 10. This time the learning ends up in a suboptimal local optimum of the cost function. One

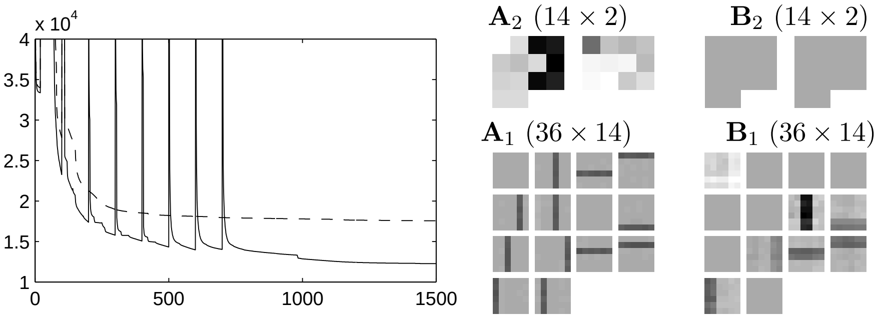


Figure 10: *Left:* Cost function plotted against the number of learning sweeps. Solid curve is the main experiment and the dashed curve is the comparison experiment. The peaks appear when nodes are added. *Right:* The resulting weights in the comparison experiment are plotted like in Fig-

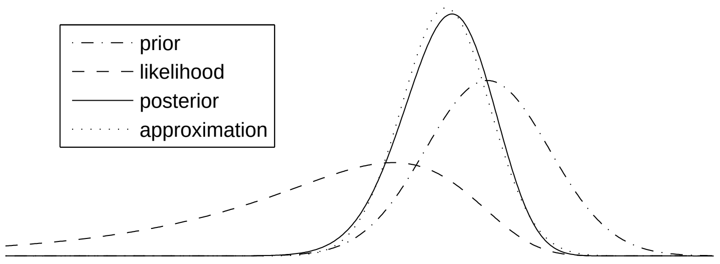


Figure 11: A typical example illustrating the posterior approximation of a variance source.

of the bars was not found (second horizontal bar from the bottom), some were mixed up in a same source (most variance bars share a source with a regular bar), fourth vertical bar from the left appears twice, and one of the sources just suppresses variance everywhere. The resulting cost function (5) is worse by 5292 compared to the main experiment. The ratio of the model evidences is thus roughly exp(5292).

Figure 11 illustrates the formation of the posterior distribution of a typical single variable. It is the first component of the variance source $\mathbf{u}_1(1)$ in the comparison experiment. The prior means here the distribution given its parents (especially $\mathbf{s}_2(1)$ and $\mathbf{B}_1$) and the likelihood means the potential given its children (the first component of $\mathbf{x}(1)$). Assuming the posteriors of other variables accurate, we can plot the true posterior of this variable and compare it to the Gaussian posterior approximation. Their difference is only 0.007 measured by Kullback-Leibler divergence.

7.2 Missing values in speech spectra

In hierarchical nonlinear factor analysis (HNFA) [76], there are a number of layers of Gaussian variables, the bottom-most layer corresponding to the data. There is a nonlinearity and a linear mixture mapping from each layer to all the layers below it.

HNFA resembles the model structure in Section 5.3. The model structure is depicted in the left subfigure of Fig. 12. Model equations are

$$\mathbf{h}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}_h(t) \tag{51}$$

$$\mathbf{x}(t) = \mathbf{B}\boldsymbol{\phi}[\mathbf{h}(t)] + \mathbf{C}\mathbf{s}(t) + \mathbf{b} + \mathbf{n}_x(t)\,, \tag{52}$$

where $\mathbf{n}_h(t)$ and $\mathbf{n}_x(t)$ are Gaussian noise terms and the nonlinearity $\phi(\xi) = \exp(-\xi^2)$ again operates on each element of its argument vector separately. Note that we have included a short-cut mapping $\mathbf{C}$ from sources to observations. This means that hidden nodes only need to model the deviations from linearity.

HNFA is compared against three other methods. Factor analysis (FA) is a linear method described in Section 5.2. It is a special case of HNFA where the dimensionality of $\mathbf{h}(t)$ is zero. Nonlinear factor analysis (NFA) [44, 32] differs from HNFA in that it does not use mediating variables $\mathbf{h}(t)$:

$$\mathbf{x}(t) = \mathbf{B}\tanh[\mathbf{A}\mathbf{s}(t) + \mathbf{a}] + \mathbf{b} + \mathbf{n}_x(t). \tag{53}$$

Note that NFA has multiple computational paths between $\mathbf{s}(t)$ and $\mathbf{x}(t)$, which leads to a higher computational complexity compared to HNFA.

The self-organising map SOM [42] differs most from the other methods. A rectangular map has a number of map units with associated model vectors that are points in the data space. Each data point is matched to the closest map unit. The model vectors of the best-matching unit and its neighbours in the map are moved slightly towards the data point. See [42, 24] for details.

The data set consisted of speech spectrograms from several Finnish subjects. Short term spectra were windowed to 30 dimensions with a standard preprocessing procedure for speech recognition. It is clear that a dynamic source model would give better reconstructions, but in this case the temporal information was left out to ease the comparison of the models. Half of the about 5000 samples were used as test data with some missing values. Missing values were set in four different ways to measure different properties of the algorithms (Figure 13):

1. 38 percent of the values are set to miss randomly in 4×4 patches. (Right subfigure of Figure 12)
2. Training and testing sets are randomly permuted before setting missing values in 4×4 patches as in Setting 1.
3. 10 percent of the values are set to miss randomly independent of any neighbours. This is an easier setting, since simple smoothing using nearby values would give fine reconstructions.

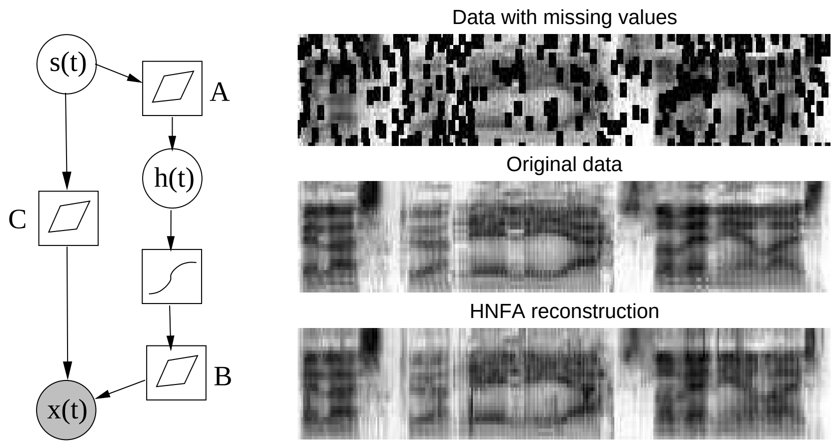


Figure 12: *Left:* The model structure for hierarchical nonlinear factor analysis (HNFA). *Right:* Some speech data with and without missing values (Setting 1) and the reconstruction given by HNFA.

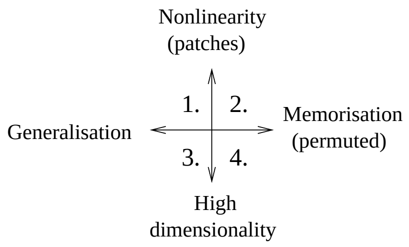


Figure 13: Four different experimental settings with the speech data used for measuring different properties of the algorithms.

4. Training and testing sets are permuted and 10 percent of the values are set to miss independently of any neighbours.

We tried to optimise each method and in the following, we describe how we got the best results. The self-organising map was run using the SOM Toolbox [79] with long learning time, 2500 map units and random initialisations. In other methods, the optimisation was based on minimising the cost function or its approximation. NFA was learned for 5000 sweeps through data using a Matlab implementation. Varying number of sources were tried out and the best ones were used as the result. The optimal number of sources was around 12 to 15 and the size used for the hidden layer was 30. A large enough number should do, since the algorithm can effectively prune out parts that are not needed.

In factor analysis (FA), the number of sources was 28. In hierarchical nonlinear factor analysis (HNFA), the number of sources at the top layer was varied and the best runs according to the cost function were selected. In those runs, the size of the top layer varied from 6 to 12 and the size of the middle layer, which is determined during learning, turned out to vary from 12 to 30. HNFA was run for 5000 sweeps through data. Each experiment with NFA or HNFA took about 8 hours of processor time, while FA and SOM were faster.

Several runs were conducted with different random initialisations but with the same data and the same missing value pattern for each setting and for each method. The number of runs in each cell is about 30 for HNFA, 4 for NFA and 20 for the SOM. FA always converges to the same solution. The mean and the standard deviation of the mean square reconstruction error are:

	FA	HNFA	NFA	SOM
Setting 1	1.87	1.80 ± 0.03	1.74 ± 0.02	1.69 ± 0.02
Setting 2	1.85	1.78 ± 0.03	1.71 ± 0.01	1.55 ± 0.01
Setting 3	0.57	$0.55 \pm .005$	$0.56 \pm .002$	0.86 ± 0.01
Setting 4	0.58	$0.55 \pm .008$	$0.58 \pm .004$	0.87 ± 0.01

The order of results of the Setting 1 follow our expectations on the nonlinearity of the models. The SOM with highest nonlinearity gives the best reconstructions, while NFA, HNFA and finally FA follow in that order. The results of HNFA vary the most - there is potential to develop better learning schemes to find better solutions more often. The sources $\mathbf{h}(t)$ of the hidden layer did not only emulate computation nodes, but they were also active themselves. Avoiding this situation during learning could help to find more nonlinear and thus perhaps better solutions.

In the Setting 2, due to the permutation, the test set contains vectors very similar to some in the training set. Therefore, generalisation is not as important as in the Setting 1. The SOM is able to memorise details corresponding to individual samples better due to its high number of parameters. Compared to the Setting 1, SOM benefits a lot and makes clearly the best reconstructions, while the others benefit only marginally.

The Settings 3 and 4, which require accurate expressive power in high dimensionality, turned out not to differ from each other much. The basic SOM has only two intrinsic dimensions[4] and therefore it was clearly poorer in accuracy. Nonlinear effects were not important in these settings, since HNFA and NFA were only marginally better than FA. HNFA was better than NFA perhaps because it had more latent variables when counting both $\mathbf{s}(t)$ and $\mathbf{h}(t)$.

To conclude, HNFA lies between FA and NFA in performance. HNFA is applicable to high dimensional problems and the middle layer can model

[4]Higher dimensional SOMs become quickly intractable due to exponential number of parameters.

part of the nonlinearity without increasing the computational complexity dramatically. FA is better than SOM when expressivity in high dimensions is important, but SOM is better when nonlinear effects are more important. The extensions of FA, NFA and HNFA, expectedly performed better than FA in each setting. It may be possible to enhance the performance of NFA and HNFA by new learning schemes whereas especially FA is already at its limits. On the other hand, FA is best if low computational complexity is the determining factor.

7.3 Variance model of image sequences

In this section an experiment with a dynamical model for variances applied to image sequence analysis is reported. The motivation behind modelling variances is that in many natural signals, there exists higher order dependencies which are well characterised by correlated variances of the signals [59]. Hence we postulate that we should be able to better catch the dynamics of a video sequence by modelling the variances of the features instead of the features themselves. This indeed is the case as will be shown.

The model considered can be summarised by the following set of equations:

$$\begin{aligned}\mathbf{x}(t) &\sim \mathcal{N}(\mathbf{A}\mathbf{s}(t), \mathrm{diag}(\exp[-\mathbf{v}_x]))\\ \mathbf{s}(t) &\sim \mathcal{N}(\mathbf{s}(t-1), \mathrm{diag}(\exp[-\mathbf{u}(t)]))\\ \mathbf{u}(t) &\sim \mathcal{N}(\mathbf{B}\mathbf{u}(t-1), \mathrm{diag}(\exp[-\mathbf{v}_u]))\end{aligned}$$

We will use the acronym DynVar in referring to this model. The linear mapping $\mathbf{A}$ from sources $\mathbf{s}(t)$ to observations $\mathbf{x}(t)$ is constrained to be sparse by assigning each source a circular region on the image patch outside of which no connections are allowed. These regions are still highly overlapping. The variances $\mathbf{u}(t)$ of the innovation process of the sources have a linear dynamical model. It should be noted that modelling the variances of the sources in this manner is impossible if one is restricted to use conjugate priors.

The sparsity of $\mathbf{A}$ is crucial as the computational complexity of the learning algorithm depends on the number of connections from $\mathbf{s}(t)$ to $\mathbf{x}(t)$. The same goal could have been reached with a different kind of approach as well. Instead of constraining the mapping to be sparse from the very beginning of learning it could have been allowed to be full for a number of iterations and only after that pruned based on the cost function as explained in Section 6.2. But as the basis for image sequences tends to get sparse anyway, it is a waste of computational resources to wait while most of the weights in the linear mapping tend to zero.

For comparison purposes, we postulate another model where the dynamical relations are sought directly between the sources leading to the following model equations:

$$\begin{aligned}\mathbf{x}(t) &\sim \mathcal{N}(\mathbf{A}\mathbf{s}(t), \mathrm{diag}(\exp[-\mathbf{v}_x])) \\ \mathbf{s}(t) &\sim \mathcal{N}(\mathbf{B}\mathbf{s}(t-1), \mathrm{diag}(\exp[-\mathbf{u}(t)]))\end{aligned}$$

We shall refer to this model as DynSrc.

The data $\mathbf{x}(t)$ was a video image sequence [78] of dimensions $16 \times 16 \times 4000$. That is, the data consisted of 4000 subsequent digital images of the size 16×16. A part of the data set is shown in Figure 14.

Both models were learned by iterating the learning algorithm 2000 times at which stage a sufficient convergence was attained. The first hint of the superiority of the DynVar model was provided by the difference of the cost between the models which was 28 bits/frame (for the coding interpretation, see [31]). To further evaluate the performance of the models, we considered a simple prediction task where the next frame was predicted based on the previous ones. The predictive distributions, $p(\mathbf{x}(t+1)|\mathbf{x}(1), ..., \mathbf{x}(t))$, for the models can be approximately computed based on the posterior approximation. The means of the predictive distributions are very similar for both of the models. Figure 15 shows the means of the DynVar model for the same sequence as in Figure 14. The means themselves are not very interesting, since they mainly reflect the situation in the previous frame. However, the DynVar model provides also a rich model for the variances. The standard deviations of its predictive distribution are shown in Figure 16. White stands for a large variance and black for a small one. Clearly, the model is able to increase the predicted variance in the area of high motion activity and hence provide better predictions. We can offer quantitative support for this claim by computing the predictive perplexities for the models. Predictive perplexity is widely used in language modelling and it is defined as

$$\mathrm{perplexity}(t) = \exp\left\{-\frac{1}{256}\sum_{i=1}^{256} \log p(x_i(t+1)|\mathbf{x}(1), ..., \mathbf{x}(t))\right\}.$$

The predictive perplexities for the same sequence as in Figure 14 are shown in Figure 17. Naturally the predictions get worse when there is movement in the video. However, DynVar model is able to handle it much better than the compared DynSrc model. The same difference can also be directly read by comparing the cost functions (3).

The possible applications for a model of image sequences include video compression, motion detection, early stages of computer vision, and making hypotheses on biological vision.

8 Discussion

One of the distinctive factors between different Bayesian approaches is the type of posterior approximation. We have concentrated on large unsuper-

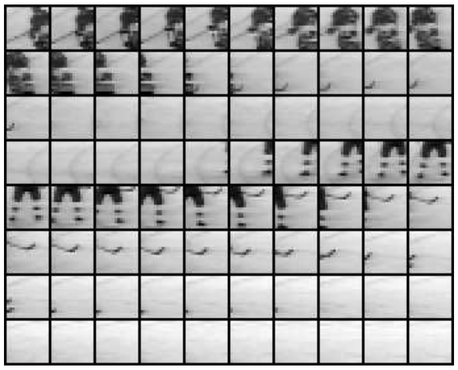


Figure 14: A sequence of 80 frames from the data used in the experiment.

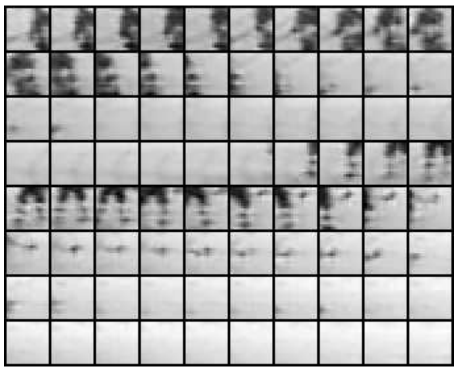


Figure 15: The means of the predictive distribution for the DynVar model.

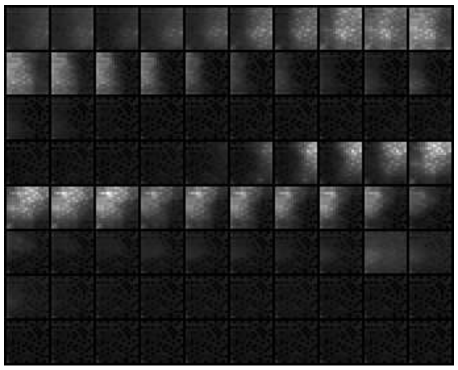


Figure 16: The standard deviations of the predictive distribution for the DynVar model.

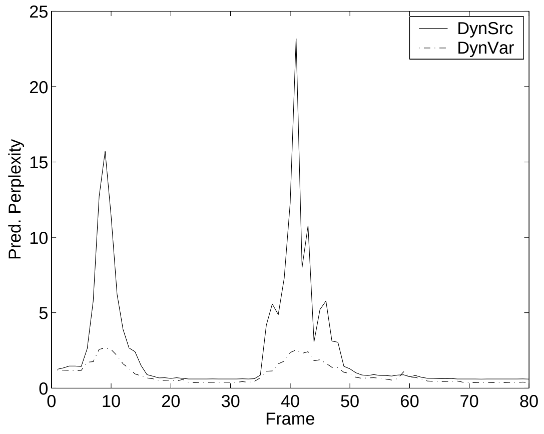


Figure 17: Predictive perplexities.

vised learning tasks, where point estimates are too prone to overfitting and sampling used in MCMC methods and particle filters, is often too slow. The problems tackled with particle filters in [14] vary from 1 to 10 in dimensionality, whereas the latent space in Section 7.3 is 128 dimensional. The variational Bayesian learning seems to provide a good compromise between point estimates and sampling methods.

Often the posterior distribution consists of clusters or solution modes. It depends on the posterior approximation again, whether only one of the clusters, or all of them are modelled. In our case, the expectation in $\mathcal{J}_{\mathrm{KL}}(q \parallel p)$ is taken over the approximate distribution q, which in practice leads to modelling a single mode. In expectation propagation [52], the Kullback-Leibler divergence is formed differently, leading to modelling of all the modes. Also, sampling is supposed to take place in all the modes. For the purpose of finding a single good representative of the posterior probability mass, the first approach should be better. In fact, the expectation over the true posterior, also known as the Bayes estimate, is often degenerate due to symmetry. For instance in factor analysis type models, the posterior is symmetric to the permutation of factors. The number of permutations also gives a hint of the infeasibility of accurately modelling all the modes in a high-dimensional problem. Perhaps it would be best to find one mode for the parameters, but all modes for the time-dependent variables when feasible.

The variational Bayesian methods vary further depending on the posterior approximation. In this paper, all variables are assumed to be independent a posteriori. We have chosen to model individual distributions as Gaussians. Often different conjugate distributions are used instead, for instance, the variance of a Gaussian variable is modelled with a Gamma distribution. Conjugate distributions are accurate and in some sense practical, but by restricting to Gaussians, the nodes can be connected more freely allowing for example hierarchical modelling of variances. It should be noted that the effect of assuming independencies is far more significant compared to the effect of approximations in modelling individual distributions.

The scope of this paper has been restricted to models which can be learned using purely local computations. This is possible, if the parents of each node are independent a posteriori. This can be accomplished by using a factorial posterior approximation and by not allowing multiple computational paths between variables. Purely local computations result in a computational complexity that is linear w.r.t. the number of connections in the model. In small models, one could afford to take all the dependencies into account. In larger models, it might be desirable to model posterior dependencies within disjoint groups of variables, but to assume the groups statistically independent, as is done by [81].

According to our experience, almost maximally factorial posterior pdf approximation $q(\boldsymbol{\theta})$ suffices in many cases. It seems that a good model structure is usually more important than a good approximation of the posterior pdf of the model. Therefore the available computation time is often better invested in a larger model using a simple posterior approximation. In any case, density estimates of continuous valued latent variables offer an important advantage over point estimates, because they are robust against overfitting and provide a cost function suitable for learning model structures. With variational Bayesian learning employing a factorial posterior pdf approximation $q(\boldsymbol{\theta})$ the density estimates are almost as efficient as point estimates. Moreover, latent variable models often exhibit rotational and other invariances which variational Bayesian learning can utilise by choosing a solution where the factorial approximation is most accurate.

The basic algorithm for learning and inference is based on updating a variable at a time while keeping other variables fixed. It has the benefits of being completely local and guaranteed to converge. A drawback is that the flow of information through time can be slow while performing inference in a dynamical model. There are alternative inference algorithms, where updates are carried out in forward and backward sweeps. These include particle smoothing [14], extended Kalman smoothing [1], and expectation propagation [52]. When the model needs to be learned at the same time, one needs to iterate a lot anyway, so the variational Bayesian algorithm that makes small but consistent improvements at every sweep might be preferable.

It is an important design choice that each node is updated while keeping the other nodes fixed. If new node types are added later on, there is no need to change the global learning algorithm, but it suffices to design an update rule for the new node type. Also, there is an option to update some nodes more often than others. When different parameters are coupled and cyclic updating is slow, it can be sped up by line search as described by [33]. Note that all the updates are done in order to minimise a global cost function, that is, a cost function over all the variables. Expectation propagation [52] updates one approximation at a time, too. An important difference is that there updates are done using a local cost function, i.e. the local approximation is fitted to the local true posterior assuming that the rest of the approximation is accurate. This is the reason why expectation propagation may diverge.

Large nonlinear problems often have numerous suboptimal local solutions that should be avoided. We have used many tricks to avoid them, as discussed in Section 6.2. It depends on the application which tricks work best. It is an important aspect of future work to make the procedure as simple as possible for the user.

9 Conclusions

In this paper, we have introduced standardised nodes (blocks) for constructing generative latent variable models. These nodes include a Gaussian node, addition, multiplication, a nonlinearity following directly a Gaussian node, and a delay node. The nodes have been designed so that they fit together, allowing construction of many types of latent variable models, including both known and novel structures. Constructing new prototype models is rapid since the user does not need to take care of the learning formulas. The nodes have been implemented in an open source software package called the Bayes Blocks [72].

The models built from these blocks are taught using variational Bayesian (ensemble) learning. This learning method essentially uses as its cost function the Kullback-Leibler information between the true posterior density and its approximation. The cost function is used for updating the unknown variables in the model, but it also allows optimisation of the number of nodes in the chosen model type. By using a factorial posterior density approximation, all the required computations can be carried out locally by propagating means, variances, and expected exponentials instead of full distributions. In this way, one can achieve a linear computational complexity with respect to the number of connections in the chosen model. However, initialisation to avoid premature pruning of nodes and local minima require special attention in each application for achieving good results.

In this paper, we have tested the introduced method experimentally in three separate unsupervised learning problems with different types of models. The results demonstrate the good performance and usefulness of the method. First, hierarchical nonlinear factor analysis (HNFA) with variance modelling was applied to an extension of the bars problem. The presented algorithm could find a model that is essentially the same as the complicated way in which the data were generated. Secondly, HNFA was used to reconstruct missing values in speech spectra. The results were consistently better than with linear factor analysis, and were generally best in cases requiring accurate representation in high dimensionality. The third experiment was carried out using real-world video image data. We compared the linear dynamical model for the means and for the variances of the sources. The results demonstrate that finding strong dependencies between different sources was considerably easier when the variances were modelled, too.

Acknowledgments

We thank Antti Honkela, Tomas Östman, and Alexander Ilin for work on Bayes blocks software library and for useful comments, and Hans van Hateren for supplying the video data [78] used in the experiments. This research has been funded by the European Commission project BLISS and the Finnish Center of Excellence Programme (2000-2005) under the project New Information Processing Principles.

APPENDICES

A Updating $q(s)$ for the Gaussian node

Here we show how to minimise the function

$$\mathcal{C}(m,v) = Mm + V[(m-m_0)^2 + v] + E\exp(m + v/2) - \frac{1}{2}\ln v, \qquad (54)$$

where M,V,E, and m_0 are scalar constants. A unique solution exists when $V > 0$ and $E \geq 0$. This problem occurs when a Gaussian posterior with mean m and variance v is fitted to a probability distribution whose logarithm has both a quadratic and exponential part resulting from Gaussian prior and log-Gamma likelihoods, respectively, and Kullback-Leibler divergence is used as the measure of the misfit.

In the special case $E = 0$, the minimum of $\mathcal{C}(m,v)$ can be found analytically and it is $m = m_0 - \frac{M}{2V}$, $v = \frac{1}{2V}$. In other cases where $E > 0$, minimisation is performed iteratively. At each iteration, one Newton iteration for the mean m and one fixed-point iteration for the variance v are carried out as explained in more detail in the following.

A.1 Newton iteration for the mean m

The Newton iteration for m is obtained by

$$\begin{aligned} m_{i+1} &= m_i - \frac{\partial \mathcal{C}(m_i, v_i)/\partial m_i}{\partial^2 \mathcal{C}(m_i, v_i)/\partial m_i^2} \\ &= m_i - \frac{M + 2V(m_i - m_0) + E \exp(m + v/2)}{2V + E \exp(m + v/2)}. \end{aligned} \tag{55}$$

The Newton iteration converges in one step if the second derivative remains constant. The step is too short if the second derivative decreases and too long if the second derivative increases. For stability, it is better to take too short than too long steps.

In this case, the second derivative always decreases if the mean m decreases and vice versa. For stability it is therefore useful to restrict the growth of m because it is consistently over-estimated.

A.2 Fixed-point iteration for the variance v

A simple fixed-point iteration rule is obtained for the variance v by solving the zero of the derivative:

$$\begin{aligned} 0 &= \frac{\partial \mathcal{C}(m, v)}{\partial v} = V + \frac{E}{2} \exp(m + v/2) - \frac{1}{2v} \Leftrightarrow \\ v &= \frac{1}{2V + E \exp(m + v/2)} \overset{def}{=} g(v) \end{aligned} \tag{56}$$

$$v_{i+1} = g(v_i) \tag{57}$$

In general, fixed-point iterations are stable around the solution v_{opt} if $|g'(v_{\text{opt}})| < 1$ and converge best when the derivative $g'(v_{\text{opt}})$ is near zero. In our case $g'(v_i)$ is always negative and can be less than -1. In this case the solution can be an unstable fixed-point. This can be avoided by taking a weighted average of (57) and a trivial iteration $v_{i+1} = v_i$:

$$v_{i+1} = \frac{\xi(v_i) g(v_i) + v_i}{\xi(v_i) + 1} \overset{def}{=} f(v_i) \tag{58}$$

The weight ξ should be such that the derivative of f is close to zero at the optimal solution v_{opt} which is achieved exactly when $\xi(v_{\text{opt}}) = -g'(v_{\text{opt}})$.

It holds

$$\begin{aligned} & g'(v) = -\frac{(E/2) \exp(m + v/2)}{[2V + E \exp(m + v/2)]^2} = g^2(v) \left[V - \frac{1}{2g(v)} \right] = g(v) \left[V g(v) - \frac{1}{2} \right] \Rightarrow \\ & g'(v_{\text{opt}}) = v_{\text{opt}} \left[V v_{\text{opt}} - \frac{1}{2} \right] \Rightarrow \xi(v_{\text{opt}}) = v_{\text{opt}} \left[\frac{1}{2} - V v_{\text{opt}} \right] \end{aligned} \tag{59}$$

The last steps follow from the fact that $v_{\mathrm{opt}} = g(v_{\mathrm{opt}})$ and from the requirement that $f'(v_{\mathrm{opt}}) = 0$. We can assume that v is close to v_{opt} and use

$$\xi(v) = v \left[\frac{1}{2} - V v_{\mathrm{opt}} \right] . \tag{60}$$

Note that the iteration (57) can only yield estimates with $0 < v_{i+1} < 1/2V$ which means that $\xi(v_{i+1}) > 0$. Therefore the use of ξ always shortens the step taken in (58). If the initial estimate $v_0 > 1/2V$, we can set it to $v_0 = 1/2V$.

A.3 Summary of the updating method for $q(s)$

1. Set $v_0 \leftarrow \min(v_0, 1/2V)$.
2. Iterate:
   (a) Solve the new estimate of the mean m from Eq. (55) under the restriction that the maximum step is 4;
   (b) Solve the new estimate of the variance v from Eqs. (60) and (58) under the restriction that the maximum step is 4.
3. Stop the iteration after the corrections become small enough, being under some suitable predefined threshold value.

B Addition and multiplication nodes

Equations (20)–(24) for the addition and multiplication nodes are proven in the following section. Only the Equation (20) applies in general, the others assume that the incoming signals are independent a posteriori. That is, $q(s_1, s_2, \dots, s_n) = q(s_1)q(s_2)\dots q(s_n)$. Also the proof of the form of the cost function mostly concerns propagation through addition and multiplication nodes, so it is presented here. Finally, the formulas of propagating the gradients of the cost function w.r.t. the expectations are derived.

B.1 Expectations

Equation (20) follows directly from the linearity of the expectation operation, or can be proven analogously to the proof of Equation (23):

$$\begin{aligned} \left\langle \prod_{i=1}^{n} s_i \right\rangle &= \int \left(\prod_{i=1}^{n} s_i \right) q(s_1, s_2, \dots, s_n) d\mathbf{s} \\ &= \int \prod_{i=1}^{n} s_i q(s_i) d\mathbf{s} = \prod_{i=1}^{n} \int s_i q(s_i) ds_i = \prod_{i=1}^{n} \langle s_i \rangle . \end{aligned}$$

Equation (21) states that the variance of a sum of independent variables is the sum of their variances. This fact can be found in basic probability theory books. It can be proven with simple manipulation by using Equations (20) and (23).

Equation (22) can be proven by applying (23) to $\exp s_i$:

$$\left\langle \exp\left(\sum_{i=1}^{n} s_i\right)\right\rangle = \left\langle \prod_{i=1}^{n} \exp s_i \right\rangle = \prod_{i=1}^{n} \langle \exp s_i \rangle .$$

Equation (24) can be proven by applying Equation (23) to both s_i and s_i^2:

$$\begin{aligned}\operatorname{Var}\left\{\prod_{i=1}^{n} s_i\right\} &= \left\langle \left(\prod_{i=1}^{n} s_i\right)^2 \right\rangle - \left\langle \prod_{j=1}^{n} s_j \right\rangle^2 = \left\langle \prod_{i=1}^{n} s_i^2 \right\rangle - \left(\prod_{j=1}^{n} \langle s_j \rangle\right)^2 \\ &= \prod_{i=1}^{n} \langle s_i^2 \rangle - \prod_{j=1}^{n} \langle s_j \rangle^2 = \prod_{i=1}^{n} \Big[\langle s_i \rangle^2 + \operatorname{Var}\{s_i\}\Big] - \prod_{j=1}^{n} \langle s_j \rangle^2 .\end{aligned}$$

B.2 Form of the cost function

The form of the part of the cost function that an output of a node affects is shown to be of the form

$$\mathcal{C}_p = M\langle\cdot\rangle + V[(\langle\cdot\rangle - \langle\cdot\rangle_{\text{current}})^2 + \operatorname{Var}\{\cdot\}] + E\langle\exp\cdot\rangle + C \qquad (61)$$

where $\langle\cdot\rangle$ denotes the expectation of the quantity in question. If the output is connected directly to another variable, this can be seen from Eq. (10) by substituting

$$\begin{aligned} M &= \langle \exp v \rangle (\langle s \rangle_{\text{current}} - \langle m \rangle) \\ V &= \frac{1}{2}\langle \exp v \rangle \\ E &= 0 \\ C &= \frac{1}{2}\Big[\langle \exp v \rangle \Big(\operatorname{Var}\{m\} + \langle m \rangle^2 - \langle s \rangle^2_{\text{current}}\Big) - \langle v \rangle + \ln 2\pi\Big] .\end{aligned}$$

If the output is connected to multiple variables, the sum of the affected costs is of the same form. Now one has to prove that this form remains the same when the signals are fed through the addition and multiplication nodes. [5]

If the cost function is of the predefined form (61) for the sum $s_1 + s_2$, it has the same form for s_1, when s_2 is regarded as a constant. This can be

[5]Note that delay node only rewires connections so it does not affect the formulas.

shown using Eqs. (20), (21), and (22):

$$\begin{aligned}
\mathcal{C}_p &= M \langle s_1 + s_2 \rangle + V \left[(\langle s_1 + s_2 \rangle - \langle s_1 + s_2 \rangle_{\text{current}})^2 + \text{Var}\{s_1 + s_2\} \right] \\
&\quad + E \langle \exp(s_1 + s_2) \rangle + C \qquad (62) \\
&= M \langle s_1 \rangle + V \left[(\langle s_1 \rangle - \langle s_1 \rangle_{\text{current}})^2 + \text{Var}\{s_1\} \right] \\
&\quad + (E \langle \exp s_2 \rangle) \langle \exp s_1 \rangle + (C + M \langle s_2 \rangle + V \text{Var}\{s_2\})
\end{aligned}$$

It can also be seen from (62) that when $E = 0$ for the sum $s_1 + s_2$, it is zero for the addend s_1, that is $E' = E \langle \exp s_2 \rangle = 0$. This means that the outputs of product and nonlinear nodes can be fed through addition nodes.

If the cost function is of the predefined form (61) with $E = 0$ for the product $s_1 s_2$, it is similar for the variable s_1, when the variable s_2 is regarded as a constant. This can be shown using Eqs. (23) and (24):

$$\begin{aligned}
\mathcal{C}_p &= M \langle s_1 s_2 \rangle + V \left[(\langle s_1 s_2 \rangle - \langle s_1 s_2 \rangle_{\text{current}})^2 + \text{Var}\{s_1 s_2\} \right] + C \qquad (63) \\
&= (M \langle s_2 \rangle + 2V \text{Var}\{s_2\} \langle s_1 \rangle_{\text{current}}) \langle s_1 \rangle \\
&\quad + \left[V \left(\langle s_2 \rangle^2 + \text{Var}\{s_2\} \right) \right] \left[(\langle s_1 \rangle - \langle s_1 \rangle_{\text{current}})^2 + \text{Var}\{s_1\} \right] \\
&\quad + \left(C - V \text{Var}\{s_2\} \langle s_1 \rangle_{\text{current}}^2 \right)
\end{aligned}$$

C Updating $q(s)$ for the Gaussian node followed by a nonlinearity

A Gaussian variable has its own terms in the cost function and it affects the cost function of its children. In case there is a nonlinearity attached to it, only the latter is changed. The cost function of the children can be written in the form

$$\mathcal{C}_{\text{ch}(s),p} = M \langle f(s) \rangle + V[(\langle f(s) \rangle - \langle f(s) \rangle_{\text{current}})^2 + \text{Var}\{f(s)\}] \qquad (64)$$

where $\langle f(s) \rangle_{\text{current}}$ stands for the expectation using the current posterior estimate $q(s)$, and M and V are constants.

The posterior $q(s) = \mathcal{N}(s; \overline{s}, \widetilde{s})$ is updated to minimise the cost function. For $\widetilde{s}$ we get a fixed point iteration for the update candidate:

$$v\widetilde{s}_{new} = \left[\langle \exp v \rangle + \frac{4V \left(1 - 2\overline{s}^2 + 2\widetilde{s}\right) (\langle f(s) \rangle - \frac{M}{2V}) \langle f(s) \rangle}{(2\widetilde{s} + 1)^2} \right. \\
\left. - \frac{4V \left(1 - 4\overline{s}^2 + 4\widetilde{s}\right) \langle [f(s)]^2 \rangle}{(4\widetilde{s} + 1)^2} \right]^{-1} \qquad (65)$$

And for $\overline{s}$ we have an approximated Newton's iteration update candidate

$$\overline{s}_{new} = \overline{s} - \widetilde{s}_{new} \left[\langle \exp v \rangle \left(\overline{s} - \langle m \rangle \right) + 4V\overline{s} \left(\frac{(\langle f(s) \rangle - \frac{M}{2V}) \langle f(s) \rangle}{2\widetilde{s}+1} - \frac{\langle [f(s)]^2 \rangle}{4\widetilde{s}+1} \right) \right] \tag{66}$$

These candidates guarantee a direction, in which the cost function decreases locally. As long as the cost function is about to increase in value, the step size is halved. This guarantees the convergence to a stable point.

D Example where point estimates fail

The following example illustrates what can go wrong with point estimates. Three dimensional data vectors $\mathbf{x}(t)$ are modelled with the linear factor analysis model $\mathbf{x}(t) = \mathbf{a}s(t) + \mathbf{n}(t)$, using a scalar source signal $s(t)$ and a Gaussian noise vector $\mathbf{n}(t)$ with zero mean and parameterised variance $p(n_k) = \mathcal{N}(0, \sigma_k^2)$. Here $\mathbf{a}$ is a three-dimensional weight vector.

The weight vector $\mathbf{a}$ might get a value $\mathbf{a} = [1\ \ 0\ \ 0]^T$, while the source can just copy the values of the first dimension of $\mathbf{x}(t)$, that is, $s(t) = x_1(t)$. When the reconstruction error or the noise term is evaluated: $\mathbf{n}(t) = \mathbf{x}(t) - \mathbf{a}s(t) = [0\ \ x_2(t)\ \ x_3(t)]^T$, one can see that problems will arise with the first variance parameter σ_1^2. The likelihood goes to infinity as σ_1^2 goes to zero. The same applies to the posterior density, since it is basically just the likelihood multiplied by a finite factor.

The found model is completely useless and still, it is rated as infinitely good using point estimates. These problems are typical for models with estimates of the noise level or products. They can be sometimes avoided by fixing the noise level or using certain normalisations [4]. When the noise model is nonstationary (see Section 5.1), the problem becomes even worse, since the infinite likelihood appears if the any of the variances goes to zero.

References

[1] B. Anderson and J. Moore. *Optimal Filtering.* Prentice-Hall, Englewood Cliffs, NJ, 1979.

[2] H. Attias. Independent factor analysis. *Neural Computation*, 11(4):803–851, 1999.

[3] H. Attias. A variational Bayesian framework for graphical models. In T. L. et al., editor, *Advances in Neural Information Processing Systems 12*, pages 209–215, Cambridge, 2000. MIT Press.

[4] H. Attias. ICA, graphical models and variational methods. In S. Roberts and R. Everson, editors, *Independent Component Analysis: Principles and Practice*, pages 95–112. Cambridge University Press, 2001.

[5] D. Barber and C. Bishop. Ensemble learning in Bayesian neural networks. In C. Bishop, editor, *Neural Networks and Machine Learning*, pages 215–237. Springer, Berlin, 1998.

[6] M. Beal. *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, University of London, UK, 2003.

[7] M. Beal and Z. Ghahramani. The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures. *Bayesian Statistics 7*, pages 453–464, 2003.

[8] C. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[9] C. Bishop. Latent variable models. In M. Jordan, editor, *Learning in Graphical Models*, pages 371–403. The MIT Press, Cambridge, MA, USA, 1999.

[10] J.-F. Cardoso. Multidimensional independent component analysis. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP'98)*, pages 1941–1944, Seattle, Washington, USA, May 12–15, 1998.

[11] K. Chan, T.-W. Lee, and T. Sejnowski. Variational learning of clusters of undercomplete nonsymmetric independent components. In *Proc. Int. Conf. on Independent Component Analysis and Signal Separation (ICA2001)*, pages 492–497, San Diego, USA, 2001.

[12] R. Choudrey, W. Penny, and S. Roberts. An ensemble learning approach to independent component analysis. In *Proc. of the IEEE Workshop on Neural Networks for Signal Processing, Sydney, Australia, December 2000*, pages 435–444. IEEE Press, 2000.

[13] P. Dayan and R. Zemel. Competition and multiple cause models. *Neural Computation*, 7(3):565–579, 1995.

[14] A. Doucet, N. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.

[15] B. J. Frey and G. E. Hinton. Variational learning in nonlinear Gaussian belief networks. *Neural Computation*, 11(1):193–214, 1999.

[16] A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC Press, Boca Raton, Florida, 1995.

[17] Z. Ghahramani and M. Beal. Propagation algorithms for variational Bayesian learning. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 507–513. The MIT Press, Cambridge, MA, USA, 2001.

[18] Z. Ghahramani and G. E. Hinton. Hierarchical non-linear factor analysis and topographic maps. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 486–492. The MIT Press, Cambridge, MA, USA, 1998.

[19] Z. Ghahramani and S. Roweis. Learning nonlinear dynamical systems using an EM algorithm. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 431–437. The MIT Press, Cambridge, MA, USA, 1999.

[20] A. Gray, B. Fischer, J. Schumann, and W. Buntine. Automatic derivation of statistical algorithms: The EM family and beyond. In *Advances in Neural Information Processing Systems 15*, 2002.

[21] M. Harva. *Hierarchical Variance Models of Image Sequences.* Helsinki Univ. of Technology, Dept. of Computer Science and Eng., Espoo, Finland, March 2004. Master of Science (Dipl.Eng.) thesis. Available at `http://www.cis.hut.fi/mha`.

[22] M. Harva and A. Kabán. A variational Bayesian method for rectified factor analysis. In *Proc. 2005 IEEE International Joint Conference on Neural Networks (IJCNN 2005)*, pages 185–190, Montreal, Canada, 2005.

[23] M. Harva, T. Raiko, A. Honkela, H. Valpola, and J. Karhunen. Bayes Blocks: An implementation of the variational Bayesian building blocks framework. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, UAI 2005*, pages 259–266, Edinburgh, Scotland, July 2005.

[24] S. Haykin. *Neural Networks – A Comprehensive Foundation, 2nd ed.* Prentice-Hall, 1998.

[25] S. Haykin, editor. *Kalman Filtering and Neural Networks.* Wiley, New York, 2001.

[26] G. E. Hinton and D. van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*, pages 5–13, Santa Cruz, CA, USA, 1993.

[27] P. Højen-Sørensen, O. Winther, and L. Hansen. Mean-field approaches to independent component analysis. *Neural Computation*, 14(4):889–918, 2002.

[28] A. Honkela. Speeding up cyclic update schemes by pattern searches. In *Proc. of the 9th Int. Conf. on Neural Information Processing (ICONIP'02)*, pages 512–516, Singapore, 2002.

[29] A. Honkela, S. Harmeling, L. Lundqvist, and H. Valpola. Using kernel PCA for initialisation of variational Bayesian nonlinear blind source separation method. In C. Puntonet and A. Prieto, editors, *Proc. of the Fifth Int. Conf. on Independent Component Analysis and Blind Signal Separation (ICA 2004)*, volume 3195 of *Lecture Notes in Computer Science*, pages 790–797, Granada, Spain, 2004. Springer-Verlag, Berlin.

[30] A. Honkela, T. Östman, and R. Vigário. Empirical evidence of the linear nature of magnetoencephalograms. In *Proc. 13th European Symposium on Artificial Neural Networks (ESANN 2005)*, pages 285–290, Bruges, Belgium, 2005.

[31] A. Honkela and H. Valpola. Variational learning and bits-back coding: an information-theoretic view to Bayesian learning. *IEEE Transactions on Neural Networks*, 15(4):800–810, 2004.

[32] A. Honkela and H. Valpola. Unsupervised variational Bayesian learning of nonlinear models. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, USA, 2005. To appear.

[33] A. Honkela, H. Valpola, and J. Karhunen. Accelerating cyclic update algorithms for parameter estimation by pattern searches. *Neural Processing Letters*, 17(2):191–203, 2003.

[34] A. Hyvärinen and P. Hoyer. Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation*, 12(7):1705–1720, 2000.

[35] A. Hyvärinen and P. Hoyer. Emergence of topography and complex cell properties from natural images using extensions of ICA. In S. A. Solla, T. K. Leen, and K.-R. Mller, editors, *Advances in Neural Information Processing Systems 12*, pages 827–833. The MIT Press, Cambridge, MA, USA, 2000.

[36] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. J. Wiley, 2001.

[37] A. Ilin and H. Valpola. On the effect of the form of the posterior approximation in variational learning of ICA models. In *Proc. of the 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*, pages 915–920, Nara, Japan, 2003.

[38] A. Ilin, H. Valpola, and E. Oja. Nonlinear dynamical factor analysis for state change detection. *IEEE Trans. on Neural Networks*, 15(3):559–575, May 2003.

[39] M. Jordan, editor. *Learning in Graphical Models.* The MIT Press, Cambridge, MA, USA, 1999.

[40] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. In M. Jordan, editor, *Learning in Graphical Models*, pages 105–161. The MIT Press, Cambridge, MA, USA, 1999.

[41] M. Jordan and T. Sejnowski, editors. *Graphical Models: Foundations of Neural Computation.* The MIT Press, Cambridge, MA, USA, 2001.

[42] T. Kohonen. *Self-Organizing Maps.* Springer, 3rd, extended edition, 2001.

[43] T. Kohonen, S. Kaski, and H. Lappalainen. Self-organized formation of various invariant-feature filters in the Adaptive-Subspace SOM. *Neural Computation*, 9(6):1321–1344, 1997.

[44] H. Lappalainen and A. Honkela. Bayesian nonlinear independent component analysis by multi-layer perceptrons. In M. Girolami, editor, *Advances in Independent Component Analysis*, pages 93–121. Springer-Verlag, Berlin, 2000.

[45] H. Lappalainen and J. Miskin. Ensemble learning. In M. Girolami, editor, *Advances in Independent Component Analysis*, pages 75–92. Springer-Verlag, Berlin, 2000.

[46] D. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[47] D. MacKay. Developments in probabilistic modelling with neural networks – ensemble learning. In *Neural Networks: Artificial Intelligence and Industrial Applications. Proc. of the 3rd Annual Symposium on Neural Networks*, pages 191–198, 1995.

[48] D. MacKay. Ensemble learning for hidden Markov models. Available at `http://wol.ra.phy.cam.ac.uk/mackay/`, 1997.

[49] D. MacKay. Introduction to Monte Carlo methods. In M. Jordan, editor, *Learning in Graphical Models*, pages 175–204. The MIT Press, Cambridge, MA, USA, 1999.

[50] D. MacKay. Local minima, symmetry-breaking, and model pruning in variational free energy minimization. Available at `http://www.inference.phy.cam.ac.uk/mackay/`, 2001.

[51] D. MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2003.

[52] T. Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, UAI 2001*, pages 362–369, Seattle, Washington, USA, 2001.

[53] J. Miskin and D. MacKay. Ensemble learning for blind source separation. In S. Roberts and R. Everson, editors, *Independent Component Analysis: Principles and Practice*, pages 209–233. Cambridge University Press, 2001.

[54] K. Murphy. A variational approximation for Bayesian networks with discrete and continuous latent variables. In *Proc. of the 15th Annual Conf. on Uncertainty in Artificial Intelligence (UAI–99)*, pages 457–466, Stockholm, Sweden, 1999.

[55] K. Murphy. The Bayes net toolbox for Matlab. *Computing Science and Statistics*, 33:331–350, 2001.

[56] R. Neal. *Bayesian Learning for Neural Networks, Lecture Notes in Statistics No. 118.* Springer-Verlag, 1996.

[57] L. Nolan, M. Harva, A. Kabán, and S. Raychaudhury. A data-driven Bayesian approach to finding young stellar populations in early-type galaxies from their UV-optical spectra. *Monthly Notices of the Royal Astronomical Society*, 2005. To appear. Available at `http://www.cis.hut.fi/mha/`.

[58] H.-J. Park and T.-W. Lee. A hierarchical ICA method for unsupervised learning of nonlinear dependencies in natural images. In C. Puntonet and A. Prieto, editors, *Proc. of the 5th Int. Conf. on Independent Component Analysis and Blind Signal Separation (ICA2004)*, pages 1253–1261, Granada, Spain, 2004.

[59] L. Parra, C. Spence, and P. Sajda. Higher-order statistical properties arising from the non-stationarity of natural signals. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 786–792. The MIT Press, Cambridge, MA, USA, 2001.

[60] J. Pearl, editor. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, San Francisco, California, 1988.

[61] D.-T. Pham and J.-F. Cardoso. Blind separation of instantaneous mixtures of nonstationary sources. *IEEE Trans. on Signal Processing*, 49(9):1837–1848, 2001.

[62] T. Raiko. Partially observed values. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN'04)*, pages 2825–2830, Budapest, Hungary, 2004.

[63] T. Raiko. Nonlinear relational Markov networks with an application to the game of Go. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN 2005)*, pages 989–996, Warsaw, Poland, September 2005.

[64] T. Raiko and M. Tornio. Learning nonlinear state-space models for control. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN'05)*, pages 815–820, Montreal, Canada, 2005.

[65] T. Raiko, H. Valpola, T. Östman, and J. Karhunen. Missing values in hierarchical nonlinear factor analysis. In *Proc. of the Int. Conf. on Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP 2003)*, pages 185–189, Istanbul, Turkey, 2003.

[66] S. Roberts and R. Everson, editors. *Independent Component Analysis: Principles and Practice.* Cambridge Univ. Press, 2001.

[67] S. Roberts, E. Roussos, and R. Choudrey. Hierarchy, priors and wavelets: structure and signal modelling using ICA. *Signal Processing*, 84(2):283–297, February 2004.

[68] D. Rowe. *Multivariate Bayesian Statistics: Models for Source Separation and Signal Unmixing.* Chapman & Hall/CRC, Medical College of Wisconsin, 2003.

[69] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

[70] D. Spiegelhalter, A. Thomas, N. Best, and W. Gilks. BUGS: Bayesian inference using Gibbs sampling, version 0.50. Available at `http://www.mrc-bsu.cam.ac.uk/bugs/`, 1995.

[71] H. Valpola, M. Harva, and J. Karhunen. Hierarchical models of variance sources. *Signal Processing*, 84(2):267–282, 2004.

[72] H. Valpola, A. Honkela, M. Harva, A. Ilin, T. Raiko, and T. Östman. Bayes Blocks software library, 2003. Available at `http://www.cis.hut.fi/projects/bayes/software/`.

[73] H. Valpola, A. Honkela, and J. Karhunen. An ensemble learning approach to nonlinear dynamic blind source separation using state-space models. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN'02)*, pages 460–465, Honolulu, Hawaii, USA, 2002.

[74] H. Valpola and J. Karhunen. An unsupervised ensemble learning method for nonlinear dynamic state-space models. *Neural Computation*, 14(11):2647–2692, 2002.

[75] H. Valpola, E. Oja, A. Ilin, A. Honkela, and J. Karhunen. Nonlinear blind source separation by variational Bayesian learning. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E86-A(3):532–541, 2003.

[76] H. Valpola, T. Östman, and J. Karhunen. Nonlinear independent factor analysis by hierarchical models. In *Proc. 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*, pages 257–262, Nara, Japan, 2003.

[77] H. Valpola, T. Raiko, and J. Karhunen. Building blocks for hierarchical latent variable models. In *Proc. 3rd Int. Conf. on Independent Component Analysis and Signal Separation (ICA2001)*, pages 710–715, San Diego, USA, 2001.

[78] J. H. van Hateren and D. L. Ruderman. Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. *Proceedings of the Royal Society of London B*, 265(1412):2315–2320, 1998.

[79] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. Self-organizing map in Matlab: the SOM toolbox. In *Proceedings of the Matlab DSP Conference*, pages 35–40, Espoo, Finland, November 1999. Available at `http://www.cis.hut.fi/projects/somtoolbox/`.

[80] C. S. Wallace. Classification by minimum-message-length inference. In S. G. Aki, F. Fiala, and W. W. Koczkodaj, editors, *Advances in Computing and Information – ICCI '90*, volume 468 of *Lecture Notes in Computer Science*, pages 72–81. Springer, Berlin, 1990.

[81] J. Winn and C. M. Bishop. Variational message passing. *Journal of Machine Learning Research*, 6:661–694, April 2005.

Publication 2

T. Raiko, H. Valpola, T. Östman, and J. Karhunen. Missing Values in Hierarchical Nonlinear Factor Analysis. In the *Proceedings of the International Conference on Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP 2003)*, pp. 185–189, Istanbul, Turkey, June 26–29, 2003.

MISSING VALUES IN HIERARCHICAL NONLINEAR FACTOR ANALYSIS

Tapani Raiko, Harri Valpola, Tomas Östman and Juha Karhunen

Helsinki University of Technology, Neural Networks Research Centre
P.O. Box 5400, FIN-02015 HUT, Espoo, Finland
`firstname.lastname@hut.fi` `http://www.cis.hut.fi/projects/ica/bayes/`

ABSTRACT

The properties of hierarchical nonlinear factor analysis (HNFA) recently introduced by Valpola and others [1] are studied by reconstructing missing values. The variational Bayesian learning algorithm for HNFA has linear computational complexity and is able to infer the structure of the model in addition to estimating the parameters. To compare HNFA with other methods, we continued the experiments with speech spectrograms in [2] comparing nonlinear factor analysis (NFA) with linear factor analysis (FA) and with the self-organising map. Experiments suggest that HNFA lies between FA and NFA in handling nonlinear problems. Furthermore, HNFA gives better reconstructions than FA and it is more reliable than NFA.

1. INTRODUCTION

A typical machine learning task is to estimate a probability distribution in the data space that best corresponds to the set of real valued data vectors $\mathbf{x}(t)$ [3]. This probabilistic model is said to be generative - it can be used to generate data. Instead of finding the distributions directly, one can assume that sources $\mathbf{s}(t)$ have generated the observations $\mathbf{x}(t)$ through a (possibly) nonlinear mapping $\mathbf{f}(\cdot)$:

$$\mathbf{x}(t) = \mathbf{f}[\mathbf{s}(t)] + \mathbf{n}(t)\,, \tag{1}$$

where $\mathbf{n}(t)$ is additive noise. Principal component analysis and independent component analysis are linear examples, but we focus on nonlinear extensions.

It is difficult to visualise the situation if for instance a 10-dimensional source space is mapped to form a nonlinear manifold in a 30-dimensional data space. Therefore, some indirect measures for studying the situation are useful. We use real-world data to make the experiment setting realistic and mark parts of the data to be missing for the purpose of controlled comparison. By varying the configuration of the missing values and then comparing the quality of their reconstructions, we measure different properties of the algorithms.

This research has been funded by the European Commission project BLISS, and the Finnish Center of Excellence Programme (2000–2005) under the project New Information Processing Principles.

Generative models handle missing values in an easy and natural way. Whenever a model is found, reconstructions of the missing values are also obtained. Other methods for handling missing data are discussed in [4]. Reconstructions are used here to demonstrate the properties of hierarchical nonlinear factor analysis (HNFA) [1] by comparing it to nonlinear factor analysis (NFA) [5], linear factor analysis (FA) [6] and to the self-organising map (SOM) [7]. Similar experiments using only the latter three methods were presented in [2].

FA is similar to principal component analysis (PCA) but it has an explicit noise model. It is a basic tool that works well when nonlinear effects are not important. The mapping $\mathbf{f}(\cdot)$ is linear and the sources $\mathbf{s}(t)$ have a diagonal Gaussian distribution. Large dimensionality is not a problem. The SOM can be presented in terms of (1), although that is not the standard way. The source vector $\mathbf{s}(t)$ contains discrete map coordinates which select the active map unit. The SOM captures nonlinearities and clusters, but has difficulties with data of high intrinsic dimensionality and with generalisation.

2. VARIATIONAL BAYESIAN LEARNING FOR NONLINEAR MODELS

Variational Bayesian (VB) learning techniques are based on approximating the true posterior probability density of the unknown variables of the model by a function with a restricted form. Currently the most common technique is ensemble learning [8] where Kullback-Leibler divergence measures the misfit between the approximation and the true posterior. It has been applied to ICA and a wide variety of other models (see [1, 9] for some references).

In ensemble learning, the posterior approximation $q(\boldsymbol{\theta})$ of the unknown variables $\boldsymbol{\theta}$ is required to have a suitably factorial form $q(\boldsymbol{\theta}) = \prod_i q_i(\boldsymbol{\theta}_i)$, where $\boldsymbol{\theta}_i$ are the subsets of unknown variables. The misfit between the true posterior $p(\boldsymbol{\theta} \mid \mathbf{X})$ and its approximation $q(\boldsymbol{\theta})$ is measured by Kullback-Leibler divergence. An additional term $-\log p(\mathbf{X})$ is included to avoid calculation of the model evidence term $p(\mathbf{X}) = \int p(\mathbf{X}, \boldsymbol{\theta}) d\boldsymbol{\theta}$. The cost function is

$$\mathcal{C} = D(q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\mathbf{X})) - \log p(\mathbf{X}) = \left\langle \log \frac{q(\boldsymbol{\theta})}{p(\mathbf{X}, \boldsymbol{\theta})} \right\rangle, \tag{2}$$

where $\langle \cdot \rangle$ denotes the expectation over distribution $q(\boldsymbol{\theta})$. Note that since $D(q \parallel p) \geq 0$, it follows that the cost function provides a lower bound for $p(\mathbf{X}) \geq \exp(-\mathcal{C})$. For a more detailed discussion, see [9].

The missing values in data behave like other latent variables and are therefore handled as a part of $\boldsymbol{\theta}$ instead of $\mathbf{X}$. The posterior approximation $q(\boldsymbol{\theta})$ is estimated during the learning and it can be used as a reconstruction for the missing values. The fraction of missing values in the data does not affect computational complexity substantially.

Beal and Ghahramani [10] compare the VB method of handling incomplete data to annealed importance sampling (AIS). In their example, the variational method works more reliably and about 100 times faster than AIS. Chan et al. [11] used ICA with VB learning successfully to reconstruct missing values. A competing approach without VB by Welling and Weber [12] has an exponential complexity w.r.t. the data dimensionality. ICA can be seen as FA with a non-Gaussian source model. Instead of going into that direction, we choose to stick to the Gaussian source model and concentrate on extending the mapping to be nonlinear instead.

2.1. Nonlinear factor analysis and hierarchical nonlinear factor analysis

In [5], a nonlinear generative model (1) was estimated by ensemble learning and the method was called nonlinear factor analysis (NFA). A more recent version with an analytical cost function and a linear computational complexity, is called hierarchical nonlinear factor analysis (HNFA) [1]. In many respects HNFA is similar to NFA. The posterior approximation, for instance, was chosen to be maximally factorial for the sake of computational efficiency and the terms $q_i(\theta_i)$ were restricted to be Gaussian.

In NFA, a multi-layer perceptron (MLP) network with one hidden layer was used for modelling the nonlinear mapping $\mathbf{f}(\cdot)$:

$$\mathbf{f}(\mathbf{s}(t); \mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}) = \mathbf{A} \tanh[\mathbf{B}\mathbf{s}(t) + \mathbf{b}] + \mathbf{a}, \tag{3}$$

where $\mathbf{A}$ and $\mathbf{B}$ are weight matrices, $\mathbf{a}$ and $\mathbf{b}$ are bias vectors and the activation function tanh operates on each element separately. The key idea in HNFA is to introduce latent variables $\mathbf{h}(t)$ before the nonlinearities and thus split the mapping (3) into two parts:

$$\begin{aligned} \mathbf{h}(t) &= \mathbf{B}\mathbf{s}(t) + \mathbf{b} + \mathbf{n}_h(t) && (4) \\ \mathbf{x}(t) &= \mathbf{A}\phi[\mathbf{h}(t)] + \mathbf{C}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}_x(t), && (5) \end{aligned}$$

where $\mathbf{n}_h(t)$ and $\mathbf{n}_x(t)$ are Gaussian noise terms and the nonlinearity $\phi(\xi) = \exp(-\xi^2)$ again operates on each element separately. Note that we have included a short-cut mapping $\mathbf{C}$ from sources to observations. This means that hidden nodes only need to model the deviations from linearity.

Learning is unsupervised and thus differs in many ways from standard backpropagation. Each step in learning tries to minimise the cost function (2). In NFA, the sources are updated while keeping the mapping constant and vice versa. The computational complexity is proportional to the number of paths from sources to the data, i.e. the product of sizes of the three layers. In HNFA, all terms $q_i(\theta_i)$ of $q(\boldsymbol{\theta})$ are updated one at a time. The computational complexity is linear with the number of connections in the model and thus HNFA scales better than NFA. In both algorithms, the update steps are repeated for several thousands of times per parameter.

In NFA, neither the posterior mean nor the variance of $\mathbf{f}(\cdot)$ over $q(\boldsymbol{\theta})$ can be computed analytically. The approximation based on Taylor series expansion may be inaccurate if the posterior variance for the input of the hidden nodes grows too large. This may be the source of the instability observed in some simulations. Preliminary experiments suggest that it may be possible to fix the problem at the expense of efficiency.

In HNFA, the posterior mean and variance of the mappings in (4) and (5) have analytic expressions. This is possible at the expense of assuming independencies of the extra latent variables $\mathbf{h}(t)$ in the posterior approximation $q(\boldsymbol{\theta})$. The assumption increases the misfit between the approximated and the true posterior. Minimisation of (2) pushes the solution in a direction where the misfit would be smaller. In [13], it is shown how this can lead to suboptimal separation in linear ICA. It is difficult to analyse the situation in nonlinear models, but it can be expected that models with fewer simultaneously active hidden nodes and thus more linear mappings are favoured. This should lead to conservative estimates of the nonlinearity of the model.

Since HNFA is built from simple blocks introduced in [14], learning the structure[1] becomes easier. The

[1]By structure, we mean the sizes of the layers and the connections between the nodes. In principle, we could allow any directed acyclic graph connecting the latent and observed variables.

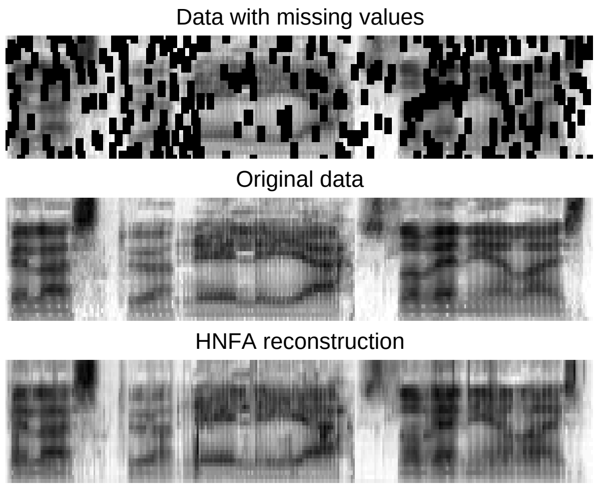


Fig. 1. Some speech data with and without missing values and the reconstruction given by HNFA.

cost function (2) relates to the model evidence $p(\mathbf{X} \mid \text{model})$ and can thus be used to compare structures. The model is built in stages starting from linear FA, i.e. HNFA without hidden nodes. See [1] for further details.

3. EXPERIMENTS

The goal is to study nonlinear models by measuring the quality of reconstructions of missing values.

The data set consists of speech spectrograms from several Finnish subjects. Short term spectra are windowed to 30 dimensions with a standard preprocessing procedure for speech recognition. It is clear that a dynamic[2] source model would give better reconstructions, but in this case the temporal information is left out to ease the comparison of the models. Half of the about 5000 samples are used as test data with some missing values. Missing values are set in four different ways to measure different properties of the algorithms (Figure 2):

1. 38 percent of the values are set to miss randomly in 4×4 patches. (Figure 1)
2. Training and testing sets are randomly permuted before setting missing values in 4×4 patches as in Setting 1.
3. 10 percent of the values are set to miss randomly independent of any neighbours. This is an easier setting, since simple smoothing using nearby values would give fine reconstructions.

[2]In [9], NFA was extended to include a model for the dynamics of the sources. A similar extension for HNFA would lead to hierarchical nonlinear dynamical factor analysis.

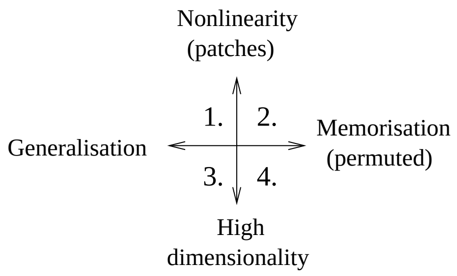


Fig. 2. Four different experimental settings with the speech data used for measuring different properties of the algorithms.

4. Training and testing sets are permuted and 10 percent of the values are set to miss independently of any neighbours.

We tried to optimise each method and in the following, we describe how we got the best results. The SOM was run using the SOM Toolbox with long learning time, 2500 map units and random initialisations. One parameter, the width of the softening kernels [2] that was used in making the reconstruction, was selected based on the results, which is not completely fair. In other methods, the optimisation was based on minimising the cost function (2) or its approximation. NFA was learned for 5000 sweeps through data using a Matlab implementation. Varying number of sources were tried out and the best ones were used as the result. The optimal number of sources was around 12 to 15 and the size used for the hidden layer was 30. A large enough number should do, since the algorithm can effectively prune out parts that are not needed. Some runs with a higher number of sources were good according to the approximation of the cost function (2), but a better approximation or a simple look at the reconstruction error of the observed data showed that those runs were actually bad. These runs and the ones that diverged were filtered out.

The details of the HNFA (and FA) implementation can be found in [1]. In FA, the number of sources was 28. In HNFA, the number of sources at the top layer was varied and the best runs according to the cost function were selected. In those runs, the size of the top layer varied from 6 to 12 and the size of the middle layer, which is determined during learning, turned out to vary from 12 to 30. HNFA was run for 5000 sweeps through data. Each run with NFA or HNFA takes about 8 hours of processor time, while FA and SOM are faster.

Several runs were conducted with different random initialisations but the same data and the same missing value pattern for each setting and for each method. The number of runs in each cell is about 30 for HNFA, 4 for NFA and 20 for the SOM. FA always converges to the same solution. The mean and the standard deviation of the mean square reconstruction error are:

	FA	HNFA	NFA	SOM
1.	1.87	1.80 ± 0.03	1.74 ± 0.02	1.69 ± 0.02
2.	1.85	1.78 ± 0.03	1.71 ± 0.01	1.55 ± 0.01
3.	0.57	$0.55 \pm .005$	$0.56 \pm .002$	0.86 ± 0.01
4.	0.58	$0.55 \pm .008$	$0.58 \pm .004$	0.87 ± 0.01

The order of results of the Setting 1 follow our expectations on the nonlinearity of the models. The SOM with highest nonlinearity gives the best reconstructions, while NFA, HNFA and finally FA follow in that order. The results of HNFA vary the most - there is potential to develop better learning schemes to find better solutions more often. The sources $\mathbf{h}(t)$ of the hidden layer did not only emulate computational nodes, but they were also active themselves. Avoiding this situation during learning could help to find more nonlinear and thus perhaps better solutions.

In the Setting 2, due to the permutation, the test set contains vectors very similar to some in the training set. Therefore, generalisation is not as important as in the Setting 1. The SOM is able to memorise details corresponding to individual samples better due to its high number of parameters. Compared to the Setting 1, SOM benefits a lot and makes clearly the best reconstructions, while the others benefit only marginally.

The Settings 3 and 4, which require accurate expressive power in high dimensionality, turned out not to differ from each other much. The basic SOM has only two intrinsic dimensions[3] and therefore it was clearly poorer in accuracy. Nonlinear effects were not important in these settings, since HNFA and NFA were only marginally better than FA. HNFA was better than NFA perhaps because it has more latent variables when counting both $\mathbf{s}(t)$ and $\mathbf{h}(t)$.

To conclude, HNFA lies between FA and NFA in performance. HNFA is applicable to high dimensional problems and the middle layer can model part of the nonlinearity without increasing the computational complexity dramatically. FA is better than the SOM when expressivity in high dimensions is important, but the SOM is better when nonlinear effects are more important. The extensions of FA, NFA and HNFA, expectedly performed better than FA in each setting. HNFA is recommended over NFA because of its reliability. It may be possible to enhance the performance of NFA and HNFA by new learning schemes whereas especially FA is already at its limits. On the other hand, FA is best if low computational complexity is the determining factor.

[3]Higher dimensional SOMs become quickly intractable due to exponential number of parameters.

4. REFERENCES

[1] H. Valpola, T. Östman, and J. Karhunen, "Nonlinear independent factor analysis by hierarchical models," in *Proc. of the 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*, 2003. To appear.

[2] T. Raiko and H. Valpola, "Missing values in nonlinear factor analysis," in *Proc. of the 8th Int. Conf. on Neural Information Processing (ICONIP'01)*, (Shanghai), pp. 822–827, 2001.

[3] C. Bishop, *Neural Networks for Pattern Recognition.* Clarendon Press, 1995.

[4] R. Little and D.B.Rubin, *Statistical Analysis with Missing Data.* J. Wiley & Sons, 1987.

[5] H. Lappalainen and A. Honkela, "Bayesian nonlinear independent component analysis by multi-layer perceptrons," in *Advances in Independent Component Analysis* (M. Girolami, ed.), pp. 93–121, Berlin: Springer-Verlag, 2000.

[6] A. Hyvärinen, J. Karhunen, and E. Oja, *Independent Component Analysis.* J. Wiley, 2001.

[7] T. Kohonen, *Self-Organizing Maps.* Springer, 3rd, extended ed., 2001.

[8] D. Barber and C. Bishop, "Ensemble learning in Bayesian neural networks," in *Neural Networks and Machine Learning* (M. Jordan, M. Kearns, and S. Solla, eds.), pp. 215–237, Berlin: Springer, 1998.

[9] H. Valpola and J. Karhunen, "An unsupervised ensemble learning method for nonlinear dynamic state-space models," *Neural Computation*, vol. 14, no. 11, pp. 2647–2692, 2002.

[10] M. Beal and Z. Ghahramani, "The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures," *Bayesian Statistics 7*, 2003. To appear.

[11] K. Chan, T.-W. Lee, and T. J. Sejnowski, "Handling missing data with variational bayesian estimation of ica," in *Proc. 9th Joint Symposium on Neural Computation*, vol. 12, (Institute for Neural Computation, Caltech), May 2002.

[12] M. Welling and M. Weber, "Independent component analysis of incomplete data," in *Proc. of the 6th Annual Joint Symposium on Neural Computation (JNSC99)*, (Pasadena), 1999.

[13] A. Ilin and H. Valpola, "On the effect of the form of the posterior approximation in variational learning of ICA models," in *Proc. of the 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*, 2003. To appear.

[14] H. Valpola, T. Raiko, and J. Karhunen, "Building blocks for hierarchical latent variable models," in *Proc. 3rd Int. Conf. on Independent Component Analysis and Signal Separation (ICA2001)*, (San Diego, USA), pp. 710–715, 2001.

Publication 3

T. Raiko. Partially Observed Values. In the *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2004)*, pp. 2825–2830, Budapest, Hungary, July 25–29, 2004.

Partially Observed Values

Tapani Raiko
Laboratory of Computer and Information Science
Helsinki University of Technology
FIN-02015 HUT, Espoo, Finland
E-mail: Tapani.Raiko@hut.fi
http://www.cis.hut.fi/projects/bayes/

***Abstract*— It is common to have both observed and missing values in data. This paper concentrates on the case where a value can be somewhere between those two ends, partially observed and partially missing. To achieve that, a method of using evidence nodes in a Bayesian network is studied. Different ways of handling inaccuracies are discussed in examples and the proposed approach is justified in the experiments with real image data. Also, a justification is given for the standard preprocessing step of adding a tiny amount of noise to the data, when a continuous-valued model is used for discrete-valued data.**

I. Introduction

Most of the data sets collected in real life are not perfect. They contain errors and missing values. There are also cases where some observations are left out on purpose, e.g. not all patients are sent to all laboratory tests. Also, some observations are more accurate or reliable than others. Usually there is some knowledge about these inaccuracies, but it is often ignored in machine learning. Fuzzy logic, on the other hand, is based on modeling inaccuracies.

Bayesian networks [1], [2] are very popular with the artificial intelligence and machine learning communities. They are graphical models [3] where nodes represent random variables and the lack of arcs represents conditional independence assumptions. A complex system is built by combining simpler parts. Traditional Bayesian networks use discrete variables but in this paper, the emphasis is on continuous variables. The experiments are run with Bayes blocks [4] that use variational Bayesian learning. They can handle missing values in a straightforward manner [5].

How to exploit the best features of the Bayesian and the fuzzy frameworks? Wald [6] proved that every admissible decision rule is a Bayes decision rule. Fuzzy logic is just a construction of heuristics, but on the other hand, fuzzy concepts are very intuitive. For instance, the distinction between the concepts *a cup* and *a bowl* is shown in [7] to be vague and context-dependent. Pearl [1] studies so called virtual evidence in Bayesian networks. It means that part of a situation is not carefully modelled but instead some evidence is summarized into virtual evidence. Virtual evidence corresponds essentially to fuzzy observations. This paper shows how virtual evidence can be used with a continuous valued model and what is it good for.

There are numerous approaches to handling missing values [8], [9] and some approaches work even in cases where the missingness of the value can depend on the actual value. But in these textbooks, a value is either observed or missing and there is no option in between. Heitjan and Rubin [10], [11] define coarse data which means that we might observe (no more and no less than) that a data value x belongs to some set, say $x \in [a, b)$. Examples include rounded and out-of-scale measurements. In this case, the value is not entirely missing, since we observe to which set it belongs to. Zhang and Honavar [12] use decision trees with partially specified data. They can specify discrete values at different levels of precision, e.g. the same shape can be described as a polygon in general or a square in specific. These hierarchies are a special case of coarse data.

Coarse data is already quite close to “fuzziness”. The gap is closed completely by using a fuzzy membership function $U(x) \in [0, 1]$ as virtual evidence for x, instead of the regular set membership restriction. I will stay in the Bayesian framework and not use fuzzy logic. Section II describes two ways of introducing fuzzy membership functions into Bayesian networks. Section III briefly reviews the variational Bayesian framework for background. Two examples that illustrate different phenomena concerning partially observed values are given in Section IV. Experiments with independent factor analysis on image data are described in Section V. Subsequently, the matters are discussed and concluded.

II. Virtual Evidence for Continuous-Valued Variables

Figure 1 shows examples of membership functions $U(x)$, which can describe different types of observations: 1) An exact observation that a person is 183 cm tall. 2) A missing observation with no knowledge of the height of this particular person. 3) A coarse observation that the person is taller than 180 cm. 4) Finally, a fuzzy observation that a person is “tall”. The common sense of peoples heights (no-one can be 3 meters tall etc.) corresponds to a model or prior experience. The question is, how to combine the knowledge given by the model to the knowledge given by the membership function.

Pearl’s virtual evidence [1] can be implemented as follows. Let us consider a Bayesian network and a single value x in it. To make x partially observed, we add a binary node e called an evidence node [13], to it (see Figure 2). The evidence node e has x as the only parent and it has no children. The conditional probability function (cpf) $p(e = 1 \mid x) = U(x)$ is the fuzzy

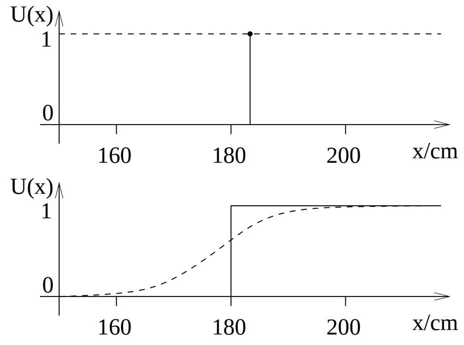


Fig. 1. Different types of observations of a person's height. Top, solid line: observed value, dashed line: missing value. Bottom, solid line: coarse observation, dashed line: fuzzy observation. All of these cases can be interpreted as partially observed values.

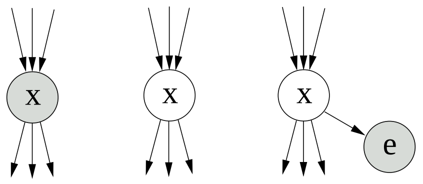


Fig. 2. The node x in a Bayesian network can be either observed (left), missing (middle) or partially observed (right). The node e is called an evidence node. A shaded node represents an observed variable and a white node represents a latent variable.

membership function $U(x)$. Now we leave x latent but observe $e = 1$. This provides evidence for x that corresponds exactly to $U(x)$ and therefore this can be called the *Evidence approach*.

The model $p(x \mid \mathcal{H}, \mathbf{X})$ for x given the model structure $\mathcal{H}$ and the rest of the data $\mathbf{X}$, is combined with the evidence given by $e = 1$. Together they form the posterior distribution

$$\begin{aligned} p(x \mid \mathcal{H}, \mathbf{X}, e = 1) &= \frac{p(x \mid \mathcal{H}, \mathbf{X}) p(e = 1 \mid x, \mathcal{H}, \mathbf{X})}{p(e = 1 \mid \mathcal{H}, \mathbf{X})} \\ &\propto p(x \mid \mathcal{H}, \mathbf{X}) p(e = 1 \mid x) \\ &= p(x \mid \mathcal{H}, \mathbf{X}) U(x). \end{aligned} \tag{1}$$

The partial observation $U(x)$ of x is thus further specified by the model. Note that the marginal likelihood $p(e = 1 \mid \mathcal{H}, \mathbf{X})$ is a constant w.r.t. x and can be thus ignored. The Evidence approach can be thought of as making a noisy observation e about x. The actual value x is then reconstructed during learning by combining prior experience $p(x \mid \mathcal{H}, \mathbf{X})$ with the evidence $U(x)$ from the noisy observation. One should be careful not to include any prior information in $U(x)$ since it would then be taken into account twice. Note also that if $U(x)$ is scaled by a constant, it still produces exactly the same evidence.

Morris et al. [14] define soft missing data by fixing a distribution over each data value: $p(x) \propto U(x)$. A Dirac delta function corresponds to a fully observed value, but unfortunately a very wide function does not approach a fully missing value as will be shown in Section IV-A. In this case, the model cannot further specify the partial observation $U(x)$, since the posterior distribution is fixed to $U(x)$. I call this the *Frozen approach*. It can be thought of as knowing that the true data is distributed in a specific way. This time, all prior information should be included in $U(x)$ but that might be difficult in practice.

Now, let us consider the continuous-valued case and a partial observation that x is probably greater than a constant c. For that, one can use the Evidence approach with a logistic membership function

$$U(x) = \frac{1}{1 + e^{-(x-c)/\alpha}}, \tag{2}$$

where α is a constant that sets the slope or fuzziness of the membership function. This can be implemented with a soft-max node [4] for e with x/α and c/α as parents. Using several different soft-max nodes combined with logical operations, one could build practically arbitrary membership functions. Note that the Frozen approach cannot handle unnormalisable membership functions such as the logistic function.

There are also other ways to produce a virtual evidence for x. One can use for instance the Gaussian evidence node [13]. A partial observation about x is that it is around x_0 with a variance σ^2. The cpf for a continuous-valued evidence node e is defined as $p(e \mid x) = N(e; x, \sigma^2)$. Observing $e = x_0$ changes the posterior distribution of x to

$$\begin{aligned} p(x \mid \mathcal{H}, \mathbf{X}, e = x_0) &= \frac{p(x \mid \mathcal{H}, \mathbf{X}) p(e = x_0 \mid x, \mathcal{H}, \mathbf{X})}{p(e = x_0)} \\ &\propto p(x \mid \mathcal{H}, \mathbf{X}) p(e = x_0 \mid x) \\ &= p(x \mid \mathcal{H}, \mathbf{X}) N(x; x_0, \sigma^2), \end{aligned} \tag{3}$$

corresponding to a Gaussian membership function $U(x) = N(x; x_0, \sigma^2)$. The last step of (3) becomes clear when noticing that the difference $e - x$ is normally distributed. The Frozen approach with a Gaussian distribution is handled simply by fixing $p(x) = N(x; x_0, \sigma^2)$.

III. Variational Bayesian Learning

Variational Bayesian learning techniques are based on approximating the true posterior probability density of the unknown variables of the model by a function with a restricted form. Currently the most common technique is ensemble learning [15], [16], [17], [18] where the Kullback-Leibler divergence measures the misfit between the approximation and the true posterior.

In ensemble learning, the posterior approximation $q(\boldsymbol{\theta})$ of the unknown variables $\boldsymbol{\theta}$ is required to have a suitably factorial form

$$q(\boldsymbol{\theta}) = \prod_i q(\boldsymbol{\theta}_i), \tag{4}$$

where $\boldsymbol{\theta}_i$ denotes a subset of the unknown variables. The misfit between the true posterior $p(\boldsymbol{\theta} \mid \mathbf{X})$ and its approximation $q(\boldsymbol{\theta})$ is measured by the Kullback-Leibler divergence. An additional term $-\ln p(\mathbf{X})$ is included to avoid calculation of the model evidence term $p(\mathbf{X}) = \int p(\mathbf{X}, \boldsymbol{\theta}) d\boldsymbol{\theta}$. The cost function then has the form [19], [15]

$$\begin{aligned}\mathcal{C} &= D(q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\mathbf{X})) - \ln p(\mathbf{X}) \\ &= \langle \ln q(\boldsymbol{\theta}) \rangle - \langle \ln p(\mathbf{X}, \boldsymbol{\theta}) \rangle ,\end{aligned} \tag{5}$$

where $\langle \cdot \rangle$ denotes expectation over the distribution $q(\boldsymbol{\theta})$. Note that since $D(q \parallel p) \geq 0$, it follows that the cost function provides a lower bound $p(\mathbf{X}) \geq \exp(-\mathcal{C})$ for the model evidence $p(\mathbf{X})$.

For each update of the posterior approximation $q(\theta_i)$, the variable θ_i requires the prior distribution $p(\theta_i \mid \text{parents})$ given by its parents and the likelihood $p(\text{children} \mid \theta_i, \text{co-parents})$ obtained from its children. The relevant part of the Kullback-Leibler divergence to be minimised is, up to a constant independent of $q(\theta_i)$

$$\mathcal{C}(q(\theta_i)) = \left\langle \ln \frac{q(\theta_i)}{p(\theta_i \mid \text{parents}) p(\text{children} \mid \theta_i, \text{co-parents})} \right\rangle . \tag{6}$$

To make it concrete, let us look at a Gaussian variable node [4] which is a basic building block for a number of models.

A Gaussian variable s has two inputs m and v and a cpf $p(s|m, v) = N(s; m, \exp(-v))$. The variance is parametrised this way because then the mean and expected exponential of v suffice for computing the cost function. It can be shown that when s, m and v are mutually independent a posteriori, i.e. $q(s, m, v) = q(s)q(m)q(v)$, $\mathcal{C}_p(q_s(s)) = -\langle \ln p(s|m, v) \rangle$ yields

$$\mathcal{C}_p(q(s)) = \frac{1}{2} \Big\{ \langle \exp v \rangle \Big[(\langle s \rangle - \langle m \rangle)^2 + \mathrm{Var}\{m\} + \\ + \mathrm{Var}\{s\} \Big] - \langle v \rangle + \ln 2\pi \Big\} . \tag{7}$$

For observed variables this is the only term in the cost function but for latent variables there is also a term $\mathcal{C}_q$ resulting from $\langle \ln q(s) \rangle$. The posterior approximation $q(s)$ is defined to be Gaussian with mean $\overline{s}$ and variance $\widetilde{s}$: $q(s) = N(s; \overline{s}, \widetilde{s})$. This yields

$$\mathcal{C}_q(q(s)) = -\frac{1}{2} \ln 2\pi e \widetilde{s} \tag{8}$$

which is the negative entropy of a Gaussian variable with variance $\widetilde{s}$. The parameters $\overline{s}$ and $\widetilde{s}$ are optimised during learning.

IV. Phenomena with Partially Observed Values

This Section gives two examples that illustrate interesting phenomena that might occur with partially observed values. Both examples concern Gaussian membership functions. In the first case, the variances are large and a comparison is done to the fully missing value. The second case shows how adding even the tiniest amount of inaccuracy to the data can make a difference by getting rid of degenerate solutions.

A. Wide Membership Functions

Figure 3 depicts an example of two-dimensional (x, y) data for factor analysis. Factor analysis is a version of principal component analysis (PCA) with a noise model. Some of the values $x(t)$ are only partially observed. Their distributions are

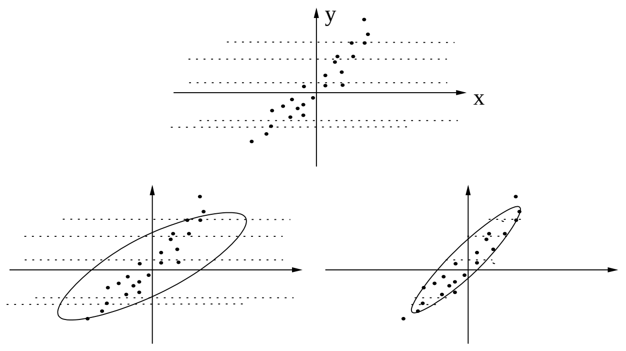


Fig. 3. Some x-values of the data are observed only partially. They are marked with dotted lines representing their confidence intervals. Top: A toy data set for a factor analysis problem. Bottom left: In the Frozen approach, the model needs to adjust to cover the distributions. Bottom right: In the Evidence approach, the partially observed values are reconstructed based on the model.

Gaussians with fairly large variances that are assumed to be known.

The Frozen approach (see Section I) assumes that the data is really distributed according to the membership function $U(x(t)) = N(x(t); \overline{x}(t), \widetilde{x}(t))$. Therefore, the model has to cover the whole distributions. In the Evidence approach, on the other hand, the posterior distribution (Eq. 1) of the partially observed values can be adjusted based on the model. Figure 3 shows the (hypothetical) situation after learning. The Frozen approach is disturbed by the partially observed values, whereas the Evidence approach reconstructs them based on the rest of the data.

When the variance $\widetilde{x}(t)$ of a Gaussian membership function goes to infinity, $U(x(t))$ is constant in any finite set. In the Evidence approach, the constant evidence corresponds to a (fully) missing value. To see what happens in the Frozen approach, one can write down the sample variance of the x-component over the data set

$$\mathrm{Var}\{x\} = \frac{1}{T-1} \sum_{t=1}^{T} \left[(\overline{x}(t) - \mathrm{E}\{x\})^2 + \widetilde{x}(t) \right] . \tag{9}$$

The model has to adjust to account for the variance in the data. When any $\widetilde{x}(t) \to \infty$, also the whole sample variance $\mathrm{Var}\{x\} \to \infty$. That is, the learning will lead to a degenerate solution in which the model for x is unreasonably wide.

B. Narrow Membership Functions

Let us think about an example of a one-dimensional mixture-of-Gaussians model for data. In case there are T data samples $x(1), \ldots, x(T)$ exactly at the same point, a Gaussian cluster with a mean $m = x(1) = \cdots = x(T)$ might specialise in those samples with a tiny variance σ^2. Ignoring the rest of the clusters and data samples, the essential likelihood factor is proportional to T/σ. When the cluster gets narrower, $\sigma \to 0$, the posterior density $p(m, \sigma \mid \mathcal{H}, \mathbf{X}) \to \infty$. That is, the solution is degenerate but it gets an infinitely good score. Note that the problem occurs even in case $T = 1$, that is, when nothing is assumed about the data.

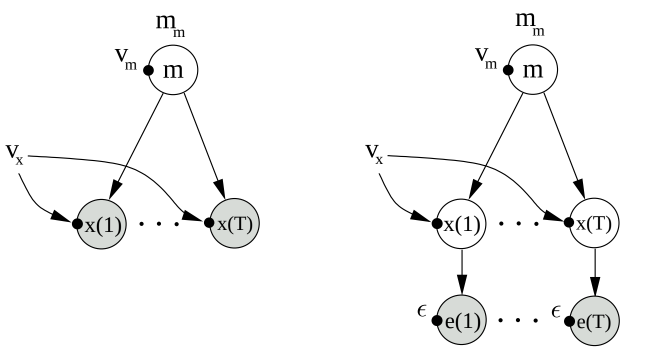


Fig. 4. A model structure representing a single Gaussian cluster with mean m containing data samples $x(1),\dots,x(T)$. On the left, the data points are fully observed and on the right, only partially observed. The dark dot at the side of a node represents the variance input.

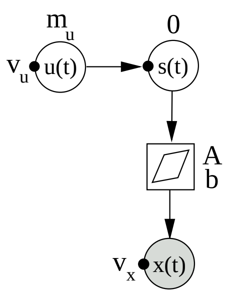


Fig. 5. The model structure used for the experiments. Each node corresponds to a matrix of variables. Variance sources u are used for making the sources s super-Gaussian. The square node represents an affine transformation with a weight matrix $\mathbf{A}$ and a bias vector $\mathbf{b}$. Hierarchical priors are hidden for clarity.

The problem is not that serious when variational Bayesian learning is used instead. Figure 4 depicts the model structure. The cluster mean has a cpf $N(m; m_m, \exp(-v_m))$ and a posterior $q(m) = N(m; \overline{m}, \widetilde{m})$. The cpfs for the data variables $x(t)$ are $N(x(t); m, \exp(-v_x))$. The essential terms of the cost function from Equations (7) and (8) are

$$\begin{aligned}\mathcal{C}(q(x,m)) &= \frac{T}{2}\left(\langle \exp v_x\rangle \widetilde{m} - \langle v_x\rangle\right) \\ &\quad + \frac{1}{2}\left(\langle \exp v_m\rangle \widetilde{m} - \ln \widetilde{m}\right). \qquad (10)\end{aligned}$$

Solving the $\widetilde{m}$ to minimize $\mathcal{C}(q(x,m))$ gives

$$\widetilde{m} = \frac{1}{T\langle \exp v_x\rangle + \langle \exp v_m\rangle} \qquad (11)$$

which is substituted back into (10) to give

$$\begin{aligned}&\mathcal{C}(q(x,m)) = \\ &\quad \frac{1}{2}\left[1 + \ln\left(T\langle \exp v_x\rangle + \langle \exp v_m\rangle\right) - T\langle v_x\rangle\right]. \qquad (12)\end{aligned}$$

In case $T > 1$, when v_x goes to infinity (corresponding to $\sigma^2 \to 0$), the cost goes to negative infinity. This means that a similar degenerate solution, that is rated infinitely good, exists in case $T > 1$.

Let us then assume that the data samples are not exactly observed. Instead, they have a Gaussian membership function with a variance $\epsilon^2 > 0$. The likelihood term at x does not change which means that maximum a posteriori learning is still prone to the same problem. Variational Bayesian learning, on the other hand, gets rid of the problem even in cases $T > 1$. Figure 4 depicts the model structure with evidence nodes. The posterior of $x(t)$ is $q(x(t)) = N(x(t); \overline{x}(t), \widetilde{x}(t))$ and the cpf of an evidence node $e(t)$ is $p(e(t) \mid x(t)) = N(e(t); x(t), \epsilon^2)$. Variances $\widetilde{m}$ and $\widetilde{x}(t)$ can be solved like in (11) and the resulting cost is similar to (12) with an additional term $(T/2)\ln(\langle \exp v_x\rangle + \epsilon^{-2})$. Now the cost approaches positive infinity when $v_x \to \infty$ and thus the degenerate solution no longer exists.

An interpretation of the situation follows. When using a point estimate for the cluster mean m, the cluster can be made infinitely narrow with no cost. In variational Bayesian learning, describing the cluster mean m with a great accuracy shows up in the cost. In case there is just one data sample $x(1)$ in the cluster, the advantage in cost is similar to the cost that went into describing m well. When $T > 1$, the advantage is T-fold and thus the degenerate solution seems infinitely good. The “happy surprise” that the data points $x(1),\dots,x(T)$ collide is as great at all levels of accuracy. But when an explicit inaccuracy of ϵ is introduced, the surprise of data points colliding is limited to the level of accuracy ϵ. An information theoretic point of view [20] to the situation is enlightening.

V. Experiments

A model structure that implements Independent factor analysis (IFA) [16], is depicted in Figure 5 and used for the experiments. The data vectors $\mathbf{x}(t)$ are assumed to be generated from unknown sources $\mathbf{s}(t)$ through an unknown linear mapping with noise

$$p(\mathbf{x}(t) \mid \cdot) = N(\mathbf{x}(t); \mathbf{A}\mathbf{s}(t) + \mathbf{b}, \mathrm{diag}(\exp(-\mathbf{v}_x))), \qquad (13)$$

where $\mathrm{diag}(\exp(-\mathbf{v}_x)))$ is a diagonal covariance matrix with values $\exp$ applied componentwise to the vector $-\mathbf{v}_x$, on the diagonal. The sources $\mathbf{s}(t)$ have a zero-mean super-Gaussian distribution generated as a Gaussian with a varying variance:

$$p(\mathbf{s}(t) \mid \mathbf{u}(t)) = N(\mathbf{s}(t); \mathbf{0}, \mathrm{diag}(\exp(-\mathbf{u}(t)))). \qquad (14)$$

The variables $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{u}(t)$ have hierarchical priors [9]. The prior of $\mathbf{A}$ is sparse (mixture of a Gaussian and a delta function at zero) and the other priors are Gaussians.

The model is initialised randomly and learned using variational Bayesian learning. The learning scheme is designed to minimise the cost function $\mathcal{C}$ in Equation (5) by iterative updates, by addition and pruning of weights, and by line search. More details can be found in [21].

The first experiment is a comparison of different ways to reconstruct corrupted values, when exact knowledge of the corruption is available. The second experiment shows a situation where the learning diverges towards a degenerate solution. Solution to avoid the problem is given.

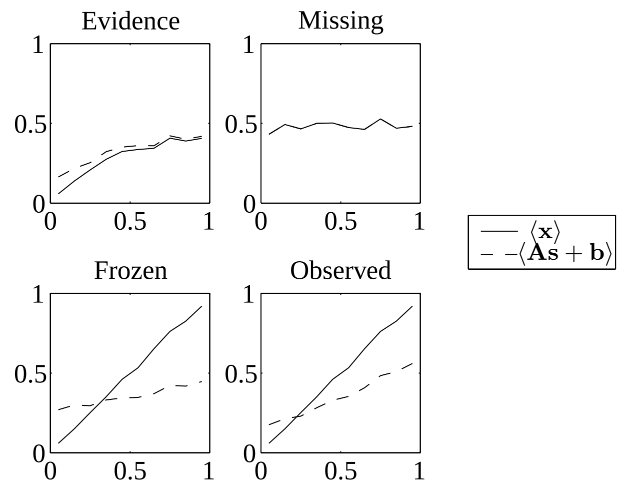


Fig. 6. Reconstruction error as a function of the amount of corruption (std).

A. Reconstruction

The first data set consists of 13 different gray-scale natural images. 1000 samples of 10-by-10-pixel patches are chosen randomly. The patches are normalised to zero mean and unit variance. Each pixel has a 10% chance of being corrupted by a Gaussian noise with a standard deviation (std) that is evenly distributed from 0 to 1. The amount of corruption is assumed to be known. That is, in addition to the data $\mathbf{x}(t)$, the stds $\mathbf{v}_e(t)$ are known. The ICA model initialised with 100 sources is learned for 1000 sweeps through the data in four different settings:

- Evidence: Evidence approach as defined in Section II. The corrupted data values ($v_{e,i}(t) > 0$) are marked missing and Gaussian evidence nodes (Eq. 3) are attached to them: $p(e_i(t) \mid x_i(t)) = N(e_i(t); x_i(t), v_{e,i}(t))$.
- Missing: Corrupted values are discarded and treated as missing values.
- Frozen: Frozen approach as defined in Section II. A Gaussian distribution with the given mean and std is fixed over each corrupted data value.
- Observed: The knowledge about corruption is discarded and values are treated as observed values.

The following table shows the root mean square errors for the reconstruction of corrupted values in different settings:

	Evidence	Missing	Frozen	Observed
$\langle \mathbf{x} \rangle$	0.31	0.48	0.57	0.57
$\langle \mathbf{As}+\mathbf{b} \rangle$	0.34	0.48	0.36	0.38

Both the expectation over the posterior distribution of data $\langle \mathbf{x}(t) \rangle$ and the conditional probability $\langle p(\mathbf{x}(t) \mid \mathbf{A}, \mathbf{s}(t), \mathbf{b}) \rangle = \langle \mathbf{As}(t)+\mathbf{b} \rangle$ are presented, because the Frozen and the Observed settings have the corrupted data directly as $\langle \mathbf{x}(t) \rangle$ (the result 0.57 is the corruption level). The same results are separated into 10 different levels of corruption and shown as curves in Figure 6. Note that the optimal constant prediction 0 gives the reconstruction error 1 since the data is normalised.

The following observations can be made:

- Evidence: As expected, the Evidence approach was the best way of reconstructing corrupted values at all corruption levels. Small corruption leads to accurate reconstructions and as the corruption level increases, the Evidence setting approaches the Missing setting.
- Missing: The data posterior $\langle \mathbf{x} \rangle$ is the same as $\langle \mathbf{As}+\mathbf{b} \rangle$. The reconstructions are independent of the corruption level since all the corrupted values were discarded. The discarded information was so important that the reconstructions were the worst.
- Frozen: Reconstructions are the second best overall. One would still need to justify when and why to use the reconstructions given by $\langle \mathbf{As}+\mathbf{b} \rangle$ and not by $\langle \mathbf{x} \rangle$. If the corruption level increases further, the reconstructions become worse than those of the Missing setting.
- Observed: Ignoring the corruption mechanism gives the second worst results. Reconstruction accuracy depends much on the corruption level.

B. Problem with noiseless data

The second data set consists of 13 different diagram-like images. They have discrete gray-scale values from 0 to 255 even though mostly they are black and white. Setting 1 has no added noise, whereas in Setting 2, a tiny amount of Gaussian noise with a standard deviation of 0.1 is added to the images. After that, figures are normalised to zero-mean and unit variance. 1000 samples of 6 by 6 image patches are chosen randomly. The same ICA-model is used, this time initialised with an over-complete basis of 50 sources.

Figure 7 shows the learning curves for the first 100 sweeps through the data. In the beginning, the two settings behave similarly, but after 45 sweeps they start to differ. After 100 sweeps, the modelled variance of the data is of the order 10^{-28} in Setting 1 and the learning is becoming unstable for numerical reasons. The same phenomenon as explained in Section IV-B is applying. The learning is diverging towards a degenerate solution that is rated infinitely good. Setting 2 is stable, even though the original difference in the two settings was very small.

The problem of a degenerate solution is often encountered when variances are modelled. As explained in Section IV-B, the problem is not as serious when using variational Bayesian learning as when using point estimates, but it still exists. The solution is to add a tiny amount of noise to the data. Whether it is done by explicitly sampling noise using a random number generator or adding the noise implicitly using either the Evidence or the Frozen approach, makes no real difference in results. Explicit sampling is usually the simplest and computationally lightest so it has become the standard.

VI. Discussion

Some real-world applications for partially observed values could be brought from the fuzzy logic community to machine learning community. Perhaps the most promising option is to find some clinical data which would contain information about

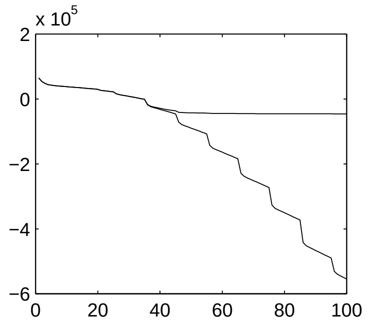


Fig. 7. The behavior of the cost function $\mathcal{C}$ during learning. The diverging lower curve corresponds to no added noise and the upper curve to a tiny amount of added noise. The regular fluctuation is expected, it reflects our learning process where every tenth iteration is done in a different manner.

the inaccuracies. Morris [14] studied speech recognition with soft missing data.

Often, it is known that the data set contains errors, but it is not known which values are erroneous. This could be modelled as evidence of evidence. The first evidence node would be left latent and its posterior distribution would tell the probability of the corresponding value to be correct or not. The second evidence node would be observed and it would give a membership function for the first evidence, and through that, some likelihood factor for the actual data value, too. It would be easier to find data for this kind of a model, since it does not require explicit knowledge of individual errors. Applications for outlier detection [22] are already well known.

Variational Bayesian learning is prone to local minima so tricks to avoid them during learning are useful. The Gaussian evidence node was first used in [13] to keep parts of the network fixed to initial values until the other parts have adapted appropriately. The width of the Gaussian evidence was increased after each iteration until the whole node was removed. The persistence of the initialisation could be thus controlled accurately.

VII. Conclusion

Partially observed values fill the gap between observed and missing values in data. A distinction is made between fixing a distribution over a data value (the Frozen approach) and getting evidence about the data value through a noisy observation (the Evidence approach). Only the Evidence approach has a missing value as a limit case. It can be implemented by adding an extra node to a Bayesian network for each partially observed value.

Experiments with natural image data and an IFA model with variational Bayesian learning show that making use of the knowledge about inaccuracies pays off. Also, a problem with applying continuous-valued models to discrete data is solved by using variational Bayesian learning combined with a tiny amount of additional noise to the data.

Acknowledgment

The author would like to thank Markus Harva, Antti Honkela, Alexander Ilin, Juha Karhunen, Erkki Oja, Jan-Hendrik Schleimer, and Harri Valpola for useful discussions. This work was supported by the Finnish Centre of Excellence Programme (2000-2005) under the project New Information Processing Principles.

References

[1] J. Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. San Francisco: Morgan Kaufmann, 1988.

[2] F. Jensen, *Bayesian Networks and Decision Graphs*. New York: Springer, 2001.

[3] K. Murphy, “An introduction to graphical models,” Intel Research Technical Report, Tech. Rep., 2001.

[4] H. Valpola, T. Raiko, and J. Karhunen, “Building blocks for hierarchical latent variable models,” in *Proc. 3rd Int. Conf. on Independent Component Analysis and Signal Separation (ICA2001)*, San Diego, USA, 2001, pp. 710–715.

[5] T. Raiko, H. Valpola, T. Östman, and J. Karhunen, “Missing values in hierarchical nonlinear factor analysis,” in *Proc. of the Int. Conf. on Artificial Neural Networks and Neural Information Processing, ICANN/ICONIP 2003*, Istanbul, Turkey, 2003, pp. 185–189.

[6] A. Wald, *Statistical Decision Functions*. New York: John Wiley & Sons, 1950.

[7] W. Labov, “The boundaries of words and their meaning,” in *New ways of analyzing variation of English*, J. Fishman, Ed. Georgetown Press, 1973, pp. 340–373.

[8] R. Little and D.B.Rubin, *Statistical Analysis with Missing Data*. J. Wiley & Sons, 1987.

[9] A. Gelman, J. Carlin, H. Stern, and D. Rubin, *Bayesian Data Analysis*. Boca Raton, Florida: Chapman & Hall/CRC Press, 1995.

[10] D. F. Heitjan and D. B. Rubin, “Inference from coarse data via multiple imputation with application to age heaping,” *Journal of the American Statistical Association*, pp. 304–314, 1990.

[11] ——, “Ignorability and coarse data,” *The Annals of Statistics*, pp. 2244–2253, 1991.

[12] J. Zhang and V. Honavar, “Learning from attribute value taxonomies and partially specified instances,” in *Proc. of the 20th International Conference on Machine Learning (ICML-2003)*, 2003, pp. 880–887.

[13] H. Valpola, T. Östman, and J. Karhunen, “Nonlinear independent factor analysis by hierarchical models,” in *Proc. 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*, Nara, Japan, 2003, pp. 257–262.

[14] A. Morris, J. Barker, and H. Bourlard, “From missing data to maybe useful data: soft data modelling for noise robust ASR,” IDIAP, IDIAP-RR 06, 2001.

[15] H. Lappalainen and J. Miskin, “Ensemble learning,” in *Advances in Independent Component Analysis*, M. Girolami, Ed. Berlin: Springer-Verlag, 2000, pp. 75–92.

[16] H. Attias, “Independent factor analysis,” *Neural Computation*, vol. 11, no. 4, pp. 803–851, 1999.

[17] J. Miskin and D. J. C. MacKay, “Ensemble learning for blind source separation,” in *Independent Component Analysis: Principles and Practice*, S. Roberts and R. Everson, Eds. Cambridge University Press, 2001, pp. 209–233.

[18] H. Valpola, E. Oja, A. Ilin, A. Honkela, and J. Karhunen, “Nonlinear blind source separation by variational Bayesian learning,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E86-A, no. 3, pp. 532–541, 2003.

[19] D. Barber and C. Bishop, “Ensemble learning in Bayesian neural networks,” in *Neural Networks and Machine Learning*, C. Bishop, Ed. Berlin: Springer, 1998, pp. 215–237.

[20] A. Honkela and H. Valpola, “Variational learning and bits-back coding: an information-theoretic view to Bayesian learning,” *IEEE Trans. on Neural Networks*, 2004, to appear.

[21] H. Valpola, M. Harva, and J. Karhunen, “Hierarchical models of variance sources,” in *Proc. 4th Int. Symp. on Independent Component Analysis and Blind Signal Separation (ICA2003)*, Nara, Japan, 2003, pp. 83–88.

[22] V. Barnett and T. Lewis, *Outliers in Statistical Data*. New York: John Wiley and Sons, 1994.

Publication 4

T. Raiko and M. Tornio. Learning Nonlinear State-Space Models for Control. In the *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2005)*, pp. 815–820, Montreal, Canada, July 31–August 4, 2005.

4

Learning Nonlinear State-Space Models for Control

Tapani Raiko and Matti Tornio
Neural Networks Research Centre
Helsinki University of Technology
P.O.Box 5400, FI-02015 TKK
Espoo, FINLAND
E-mail: tapani.raiko@hut.fi, matti.tornio@hut.fi

***Abstract*— This paper studies the learning of nonlinear state-space models for a control task. This has some advantages over traditional methods. Variational Bayesian learning provides a framework where uncertainty is explicitly taken into account and system identification can be combined with model-predictive control. Three different control schemes are used. One of them, optimistic inference control, is a novel method based directly on the probabilistic modelling. Simulations with a cart-pole swing-up task confirm that the latent state space provides a representation that is easier to predict and control than the original observation space.**

I. Introduction

Nonlinear control is difficult even in the case that the system dynamics are known. If the dynamics are not known, the traditional approach is to make a model of the dynamics (system identification) and then try to control the simulated model (nonlinear model-predictive control). The model learned from data is of course not perfect, but these imperfections are often ignored. The modern view of control sees feedback as a tool for uncertainty management [11], but managing it already in the modelling might have advantages. For instance, the controller can avoid regions where the confidence in model is not high enough [9].

The idea of studying uncertainty in control is not new. It is known that the magnitude of motor noise in human hand motion is proportional to muscle activation [10]. In control theory, the theoretical foundations are already well covered in [3]. In [14], a nonlinear state-space model is used for control. The nonlinearities are modelled using piecewise affine mappings. Parameters are estimated using the prediction error method, which is equivalent to the maximum likelihood estimate in the Bayesian framework.

Nonlinear dynamical factor analysis (NDFA) [17] is a state-of-the-art tool for finding nonlinear state-space models with variational Bayesian learning. This paper is about using NDFA for control. In NDFA, the parameters, the states, and the observations are real-valued vectors that are modelled with parametrised probability distributions. Uncertainties from noisy observations and model imperfections are thus taken explicitly into account.

Learning is extremely important for control of complex systems [2]. The proposed method involves learning in more than one way. The original NDFA is based on unsupervised learning. That is, it creates a model of the underlying dynamics by passively making observations. When used for control, though, control signals need to be selected either by following an example or by maximising a reward. The model should thus not only learn the dynamics, but also learn to help control.

The rest of the paper is structured as follows: In Section II, a nonlinear state-space model is reviewed and in Section III its use as a controller is presented. After experiments in Section IV matters are discussed and concluded.

II. Nonlinear State-Space Models

Nonlinear dynamical factor analysis (NDFA) [17] is a powerful tool for modelling the dynamics of an unknown noisy system. NDFA scales only quadratically with the dimensionality of the observation space, so it is also suitable for modelling systems with fairly high dimensionality [17].

In NDFA, the observations $\mathbf{x}(t)$ have been generated from the hidden state $\mathbf{s}(t)$ by the following generative model:

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t), \boldsymbol{\theta}_\mathbf{f}) + \mathbf{n}(t) \tag{1}$$
$$\mathbf{s}(t) = \mathbf{g}(\mathbf{s}(t-1), \boldsymbol{\theta}_\mathbf{g}) + \mathbf{m}(t), \tag{2}$$

where $\boldsymbol{\theta}$ is a vector containing the model parameters and time t is discrete. The noise terms $\mathbf{n}(t)$ and $\mathbf{m}(t)$ are assumed to be Gaussian and white. Only the observations $\mathbf{x}$ are known beforehand, and both the states $\mathbf{s}$ and the mappings $\mathbf{f}$ and $\mathbf{g}$ are learned from the data.

Multilayer perceptron (MLP) networks [6] suit well to modelling both strong and mild nonlinearities. The MLP network models for $\mathbf{f}$ and $\mathbf{g}$ are

$$\mathbf{f}(\mathbf{s}(t), \boldsymbol{\theta}_\mathbf{f}) = \mathbf{B} \tanh\left[\mathbf{A}\mathbf{s}(t) + \mathbf{a}\right] + \mathbf{b} \tag{3}$$
$$\mathbf{g}(\mathbf{s}(t), \boldsymbol{\theta}_\mathbf{g}) = \mathbf{s}(t) + \mathbf{D} \tanh\left[\mathbf{C}\mathbf{s}(t) + \mathbf{c}\right] + \mathbf{d}, \tag{4}$$

where the sigmoidal tanh nonlinearity is applied component-wise to its argument vector. The parameters $\boldsymbol{\theta}$ include: (1) the weight matrices $\mathbf{A} \dots \mathbf{D}$, the bias vectors $\mathbf{a} \dots \mathbf{d}$; (2) the parameters of the distributions of the noise signals $\mathbf{n}(t)$ and $\mathbf{m}(t)$ and the column vectors of the weight matrices; (3) the hyperparameters describing the distributions of biases and the parameters in group (2).

There are infinitely many models that can explain any given data. In Bayesian learning, all the possible explanations are averaged weighting by their posterior probability. The posterior probability $p(\mathbf{s}, \boldsymbol{\theta} \mid \mathbf{x})$ of the states and the parameters after observing the data, contains all the relevant information about them. Variational Bayesian learning is a way to approximate the posterior density by a parametric distribution $q(\mathbf{s}, \boldsymbol{\theta})$. The misfit is measured by the Kullback-Leibler divergence:

$$C_{\mathrm{KL}} = \int q(\mathbf{s}, \boldsymbol{\theta}) \log \frac{q(\mathbf{s}, \boldsymbol{\theta})}{p(\mathbf{s}, \boldsymbol{\theta} \mid \mathbf{x})} d\boldsymbol{\theta} d\mathbf{s}. \tag{5}$$

The approximation q needs to be simple for mathematical tractability and computational efficiency. Variables are assumed to depend of each other in the following way:

$$q(\mathbf{s}, \boldsymbol{\theta}) = \prod_{t=1}^{T} \prod_{i=1}^{m} q(s_i(t) \mid s_i(t-1)) \prod_{j} q(\theta_j), \tag{6}$$

where m is the dimensionality of the state space $\mathbf{s}$. Furthermore, q is assumed to be Gaussian.

Learning and inference happen by adjusting q such that the cost function C_{KL} is minimised. A good initialisation and other measures are essential because the iterative learning algorithm can easily get stuck into a local minimum of the cost function. The standard initialisation is based on principal component analysis of the data augmented with embedding. Details can be found in [17].

A. Iterated Extended Kalman Smoothing

In a typical NDFA learning phase, both the model parameters $\boldsymbol{\theta}$ and the states $\mathbf{s}$ are updated. The updating of the network weights is computationally the most expensive part of the process, so the speed of the updating of the states is of minor importance [17]. In the control schemes studied in this work, however, the model parameters can be kept fixed and only the states are inferred. As a faster alternative to the update process used in the NDFA Matlab package, extensions of Kalman filtering [7] are explored.

Kalman smoothing estimates the state of a linear Gaussian state-space model in a two-phase forward and backward pass. Extended Kalman smoothing [1] does the same for a nonlinear model by linearising the model based on the current estimate of the states and then applying linear Kalman smoothing. The process iterates between updating the states and the linearisation.

Kalman-based methods are fast because they propagate information through the whole time window in every iteration, whereas the update rules included in NDFA propagate information only one step forward and backward per iteration. Unfortunately, Kalman-based methods have no guarantee of convergence when applied to nonlinear systems. To solve this issue we used iterated extended Kalman smoothing for finding a good initialisation which was then improved by some NDFA updates.

In this work, a non-variational Kalman smoother is used. A variational Kalman smoother does exists [4], but as the Kalman smoother is used only for the initialisation of NDFA, the added complexity was not deemed worthwhile.

B. Task-Oriented Identification

When the dynamic system is controlled by a continuous-valued control signal vector $\mathbf{u}(t)$, it can be taken into account

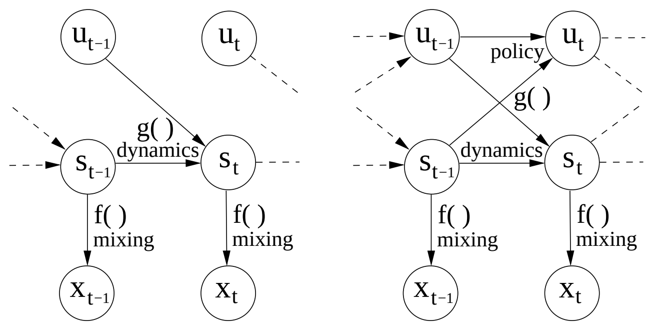


Fig. 1. Traditional model (left) and task-oriented identification (right). Traditionally, the control signals $\mathbf{u}(t)$ are coming from outside the model, but in task-oriented identification they are within the model.

by replacing the equation of dynamics (2) with one of these two options:

$$\mathbf{s}(t) = \mathbf{g}\left(\begin{bmatrix} \mathbf{u}(t-1) \\ \mathbf{s}(t-1) \end{bmatrix}, \boldsymbol{\theta}_{\mathbf{g}}\right) + \mathbf{m}(t) \tag{7}$$

$$\begin{bmatrix} \mathbf{u}(t) \\ \mathbf{s}(t) \end{bmatrix} = \mathbf{g}\left(\begin{bmatrix} \mathbf{u}(t-1) \\ \mathbf{s}(t-1) \end{bmatrix}, \boldsymbol{\theta}_{\mathbf{g}}\right) + \mathbf{m}(t). \tag{8}$$

The first one (7) assumes that the control signal is coming outside the model. The latter one (8) is called task-oriented identification because it predicts the control signals $\mathbf{u}(t)$ within the model. Figure 1 illustrates these two options.

We choose to use task-oriented identification (Eq. 8) in this paper for the following reasons. Firstly, it allows for three different control schemes described in the next section. Secondly, it creates an opportunity to learn more. The learning algorithm finds such a state space that the prediction of observations and control signals is as accurate as possible. A well-learned state space should thus make control easier. Thirdly, it is biologically motivated. Different parts of the cerebellum can be used for motor control and cognitive processing depending on where their outputs are directed [5].

III. Control Schemes

So far only passive observation and learning has been considered. Now we come to the question how the control signals (or actions) are selected. That is, given the history of observations $\ldots, \mathbf{x}(t_0-2), \mathbf{x}(t_0-1)$ and control signals $\ldots, \mathbf{u}(t_0-2), \mathbf{u}(t_0-1)$, select a good control signal $\mathbf{u}(t_0)$ at the current time t_0. Then, a new observation $\mathbf{x}(t_0)$ is made and time t_0 is increased by one. Three different control schemes and their cooperation are studied below and summarised in Table I.

A. Direct Control (DC)

In direct control schemes, the neural network itself acts as the controller. Many such schemes exists, including direct inverse control, optimal control, and feedforward control [13]. Direct control can only mimic the control done in the data that has been used for learning. It therefore requires examples of correct control aiming at the same goal.

Equation (8) provides a prediction of the control signal $\mathbf{u}(t_0)$ based on the previous control signal $\mathbf{u}(t_0-1)$ and the

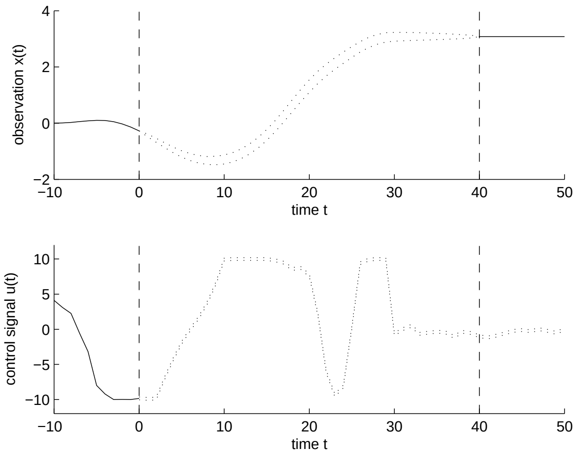


Fig. 2. Optimistic inference control (see Section III-B). The inferred observations and control signals are plotted with confidence intervals. The current time is $t_0 = 0$ and after time $t_0 + T_c = 40$, the observation $\mathbf{x}(t)$ is assumed to be at the desired level $\mathbf{r}$.

previous estimate of the hidden state $\mathbf{s}(t_0 - 1)$. The prediction mapping is called the policy in Figure 1. A control method that we simply call direct control (DC), chooses the control signal by collapsing the inferred probability distribution $q(\mathbf{u}(t_0))$ to its expected value. When the control signal $\mathbf{u}(t_0)$ is selected and the observation $\mathbf{x}(t_0)$ is made, the two probability distribution collapse and these changes affect the estimates of the states $\mathbf{s}(t)$ that are then re-inferred. This works as the error feedback mechanism.

B. Optimistic Inference Control (OIC)

Optimistic inference control (OIC) is a novel method which works as follows. Assume that after a fixed delay T_c, the desired goal is reached. That is, (some components of) the observations $\mathbf{x}$ are at the desired level $\mathbf{r}$. Given this optimistic assumption and the observations and control signals so far, infer what happens in between. Then choose the expectation of $q(\mathbf{u}(t_0))$ as before. An example situation is illustrated in Figure 2.

```
OIC in a nutshell:
Given observations $\dots, \mathbf{x}(t_0-2), \mathbf{x}(t_0-1)$ and
control signals $\dots, \mathbf{u}(t_0-2), \mathbf{u}(t_0-1)$
1:    Fix future $\mathbf{x}(t_0+T_c) = \mathbf{x}(t_0+T_c+1) = \cdots = \mathbf{r}$
2:    Infer the distribution $q(\mathbf{u}(t), \mathbf{s}(t), \mathbf{x}(t))$ for all $t$
3:    Select the mean of $q(\mathbf{u}(t_0))$ as the control signal
4:    Observe $\mathbf{x}(t_0)$ and release $\mathbf{x}(t_0+T_c)$
5: Increase $t_0$ and loop from 1
```

OIC propagates the same evidence forwards as the DC and additionally, the evidence from the desired future backwards. The inference is conceptually simple, but algorithmically difficult. The information from the future needs to flow through tens of nonlinear mappings $\mathbf{g}$ before it affects $\mathbf{u}(t_0)$.

In case there are constraints for control signals or observations, they are forced after every inference iteration. If the horizon is set too short or the goal is otherwise overoptimistic, the method becomes unreliable. Even with a realistic goal, it is not in general guaranteed that the iteration will converge to the optimal control signal, as the iteration may get stuck in a local minimum. The inferred control signals can be validated by releasing the optimistic future and re-inferring. If the future changes a lot, the control is unreliable. Note that OIC does not require goal-oriented data, because different goals can be set by changing the desired future.

TABLE I
CONTROL SCHEME SUMMARY

Scheme	Based on	Data	Speed
DC	internal MLP	task-oriented	fast
OIC	probabilistic inference	general	slow
NMPC	cost minimisation	general	slow

C. Nonlinear Model Predictive Control (NMPC)

Nonlinear model predictive control (NMPC) [13] is based on minimising a cost function J defined over a future window of fixed length T_c. For example, the quadratic difference between the predicted future observations $\mathbf{x}$ and a reference signal $\mathbf{r}$ can be used:

$$J(\mathbf{s}(t_0), \mathbf{u}(t_0), \dots, \mathbf{u}(t_0 + T_c - 1)) = \sum_{\tau=1}^{T_c} |\mathbf{x}(t_0 + \tau) - \mathbf{r}|^2 . \quad (9)$$

Then J is minimised w.r.t. the control signals $\mathbf{u}$ and the first one $\mathbf{u}(t_0)$ is executed.

In this paper, the states and observations (but not control signals) are modelled probabilistically so we actually minimise the expected cost $E_q\{J\}$. The current guess $\mathbf{u}(t_0), \dots, \mathbf{u}(t_0 + T_c - 1)$ defines a probability distribution over future states and observations. This inference can be done with a single forward pass, when ignoring the policy mapping, that is, the dependency of the state on future control signals. In this case, it makes sense to ignore the policy mapping anyway, since the future control signals do not have to follow the policy.

Minimisation of $E_q\{J\}$ is done with a certain quasi-Newton algorithm [12]. For that, the partial derivatives $\partial\mathbf{x}(t_2)/\partial\mathbf{u}(t_1)$ for all $t_0 \le t_1 < t_2 \le t_0 + T_c$ are computed efficiently based on the chain rule and dynamic programming. Details are left for future publications due to lack of space.

The use of a cost function makes NMPC very versatile. Costs for control signals and observations can be set for instance to restrict values within bounds etc. Quadratic costs such as (9) make things easy for the optimisation algorithm.

IV. EXPERIMENTS

Mechanical dynamical systems are easily understandable by people and thus illustrative as examples. We chose a simulated system to ease experimentation. To make the setting more realistic, the controllers do not have access to the simulation equations but have to adapt to control an unknown system instead.

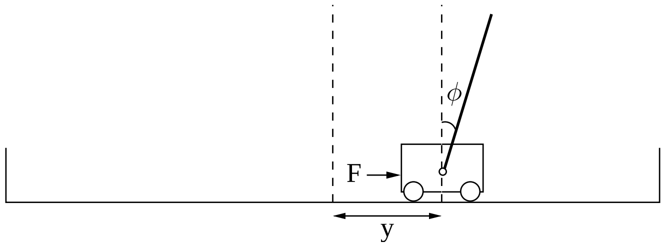


Fig. 3. The cart-pole system

A. Cart-Pole Swing-Up Task

The Cart-Pole system [8] is a classic benchmark for nonlinear control. The system consist of a pole (which acts as an inverted pendulum) attached to a cart (Figure 3). The force applied to the cart can be controlled, and the goal is to swing the pole to an upward position and stabilise it. This must be accomplished without the cart crashing into the walls of the track. Note that a linear controller cannot perform the swing-up.

The observed variables of the system are the position of the cart y, angle of the pole measured from the upward position ϕ, and their first derivatives y' and ϕ'. Control input is the force F applied to the cart. The detailed dynamics and constraints for the simulated cart-pole system can be found in [8].

A discrete system was simulated with a time step of $\Delta t = 0.05$s. The possible force was constrained between -10N and 10N, and the position between -3m and 3m. The system was initialised to a random state around $[y, y', \phi, \phi'] = [0, 0, -\pi, 0]$ with a standard deviation of 0.1 for all the observed variables.

B. Simulation

All simulations were ran with both low ($\sigma = 0.001$) and high ($\sigma = 0.1$) level of Gaussian additive observation noise. Gaussian process noise with $\sigma = 0.001$ was used in all the simulations and the training data set. For the NMPC and OIC methods the length of the control horizon was set to 40 time steps corresponding to 2 seconds of system's real time. The simulations were run for 60 time steps corresponding to 3 seconds of real time to ensure that the controller was able to stabilise the pole.

To study the benefits of using a hidden state-space in modelling the dynamics of an unknown system, a comparison model was built which used identity mapping $\mathbf{I}$ instead of an MLP $\mathbf{f}$ for the observation mapping. In practice this means replacing (1) with

$$\mathbf{x}(t) = \mathbf{s}(t) + \mathbf{n}(t). \tag{10}$$

Also, a modified version of the problem was considered, where only two observations, the location of the cart y and the angle of the pole ϕ, were available.

C. Implementation

The NDFA package version 0.9.5, the scripts for running the experiments, and the used training data are publicly available[1].

[1]http://www.cis.hut.fi/projects/bayes/software/

During the training phase for indirect methods, training data with 2500 samples was used. In [18], different reinforcement learning algorithms require from 9000 up to 2500000 samples to learn to control the cart. Most of the training data consisted of a sequence generated with semi-random control where the only goal was to ensure that the cart does not crash into the boundaries. Training data also contained some examples of hand-generated sections to better model the whole range of the observation and the dynamic mapping. The model was trained for 500000 iterations, which translates to three days of computation time. Six-dimensional state space $\mathbf{s}(t)$ was used because it resulted in a model with the lowest cost function (Eq. 5).

For the direct control method, training data consisted of 30 examples of successful swing-ups with 100 samples each. They were generated using the NMPC method with a horizon length of 40 time steps. Four-dimensional state space proved to be the best here, and the model was trained for 100000 iterations.

For all the models, the first 1000 iterations of the training were run with the embedded versions of the data to avoid bad local optima. Time-shifted versions of the observed data $\mathbf{x}(t-\tau)$, with $\tau = 1, 2, 4, 8, 16$, were used in addition to the original data.

The state $\mathbf{s}(t)$ was estimated using the iterated extended Kalman smoother. A history of five observations and control signals seemed to suffice to give a reliable estimate. The reference signal $\mathbf{r}$ was $\phi = 0$ and $\phi' = 0$ at the end of the horizon and for five observations beyond that.

To take care of the constraints in the system with NMPC, a slightly modified version of the cost function (9) was used. Out-of-bounds values of the location of the cart and the force incurred a quadratic penalty, and the full cost function is of the form

$$\begin{aligned} J_1(t_0, \mathbf{u}) = & J(t_0, \mathbf{u}) + \\ & \sum_{\tau=1}^{T_c} (\min(10, |u(t_0+\tau)|) - 10)^2 + \\ & \sum_{\tau=1}^{T_c} (\min(3, |x_y(t_0+\tau)|) - 3)^2, \end{aligned} \tag{11}$$

where $x_y(t)$ refers to the location component y of the observation vector $\mathbf{x}(t)$.

D. Simulation Results

For all the control schemes, the cart-pole simulation was run for 100 times and the number of successful swing-ups was collected. As in [8], a swing-up is considered successful if the final angle is between -0.133π and 0.133π, final angular velocity between -2rad/s and 2rad/s, and the cart has not crashed into the boundaries of the area during swing-up.

The results of all the simulations are collected in Table II. For each simulation type, the number of successful swing-ups and the number of partial successes are listed. The partial successes include all the simulation runs that at some point

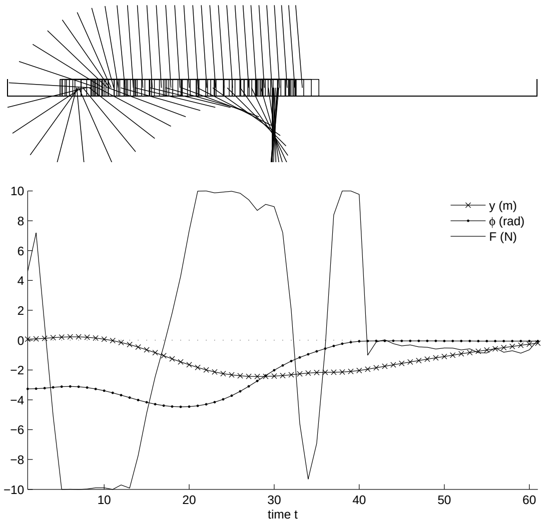


Fig. 4. Example of a successful swing-up with NMPC and low noise. The cart starts from the middle with the pole hanging down, and goes left to swing the pole up.

TABLE II

RESULTS: NUMBER OF SUCCESSFUL AND SEMI-SUCCESSFUL (IN BRACKETS) SWING-UPS WITH LOW AND HIGH NOISE LEVEL σ.

Setting	$\sigma = 0.001$		$\sigma = 0.1$	
Direct Control	14	(48)	4	(31)
Optimistic Inference Control	97	(100)	94	(98)
NMPC	100	(100)	94	(95)
NMPC (only y and ϕ observed)	14	(66)	1	(21)
NMPC ($\mathbf{f} = \mathbf{I}$)	100	(100)	70	(70)
NMPC ($\mathbf{f} = \mathbf{I}$, only y and ϕ)	0	(0)	0	(0)

reached the desired state, but possibly still failed either because the pole was not stabilised or the cart crashed into a wall.

1) Direct Control: The direct control could perform the swing-up part of the task quite well, but there were problems with stabilising the pole. Further testing is still needed to verify if the performance of the method can be improved by extra training with pole stabilising data.

2) Indirect Control: Even though there was some modelling error left in the model used with indirect control schemes, both methods performed extremely well under low noise conditions. Even with added noise, the performance was pretty satisfactory. Examples of successful swing-ups can be found in Figures 4 and 5.

3) Performance: With modern hardware (2.2 GHz AMD Opteron) the direct control typically worked in real-time with the cart-pole simulation. On average, the traditional NMPC method was about 20 times slower than real-time and OIC more than 100 times slower. The bad performance of OIC resulted from the Kalman smoothing (see Section II-A) not converging and having to switch to the slow update mode. Further optimisations to the algorithms or improvements in hardware are clearly required, before systems with fast dynamics can be controlled.

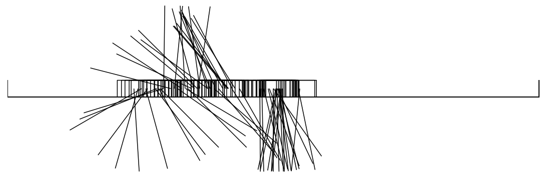


Fig. 5. Example of a successful swing-up with NMPC and high noise. The system is plotted with the observation noise included.

4) Dynamic Model Based Directly on the Observations: With the modified model using the observation space as the state space, the performance was still perfect when the noise level was low. However, with high noise level, the original model performed clearly better than the modified model.

5) Models with Fewer Observations: Even though most of the information on the speed y' and the angular velocity ϕ' can still be inferred taking into account past observations, in practice the problem of learning the dynamics of the system becomes harder and relying on past observations increases the reaction time. The model with hidden state could still perform the swing-up with some success, but a model based directly on observations could not handle the swing-up at all. This result was to be expected, as the dynamic mapping (8) alone cannot adequately describe the modified system.

6) Horizon length: Horizon length was of no great importance to the performance of the NMPC or the OIC. All horizon lengths between 30 and 45 time steps had similar performance. Horizon lengths between 25 and 30 had problems with the cart crashing to the walls. Horizons shorter than 25 time steps could not reliably perform the swing-up task because the reference signal became too unrealistic.

Very long horizons were also problematic. First of all, they increase the computational burden of the algorithm. The increase in the number of the parameters often also leads into increase in the number of local minima, which makes the optimisation problem more involved. In addition, because only an approximative model of the system is available, predictions far to the future become more unreliable. This can lead the algorithm to choose an optimisation strategy which is not feasible in practice.

V. DISCUSSION AND CONCLUSION

Three different control schemes were studied in the framework of nonlinear state-space models. Direct control is fast to use, but requires the learning of a policy mapping, which is hard to do well. Optimistic inference control is a novel method based on Bayesian inference answering the question: “Assuming success in the end, what will happen in near future?” It is based on a single probabilistic inference but unfortunately neither of the two tested inference algorithms work well with it. The third control scheme is a probabilistic version of the standard nonlinear model-predictive control, which is based on optimising control signals based on a cost function. The latter two schemes are both indirect control methods and they performed comparably well in the experiments.

A. Future Work

When learning from data, the model represents well only those phenomena that appear in the data. If the data is too uniform, the model will not become robust. In other words, one should balance between exploration and exploitation. In this paper, the data sets are generated partly by hand and all the control schemes aim at exploitation only. A good starting point for taking exploration into account is in [16].

For direct control, the model was learned using examples of control with a single goal in mind. It is straightforward to generalise this into a situation with a selection of different goals. The dynamics of the system stays the same regardless of the goal and only the policy mapping (see Figure 1) needs to be changed for each goal.

The direct and indirect control methods can be used together. One can use the data produced by indirect control methods for learning the direct controller. This can be done even offline, that is, simulating the estimated model and sampling observations from their predicted distributions. This can be compared to dreaming. The enhancement of the task-oriented identification (policy mapping) in turn helps the indirect methods, too. This idea is comparable to temporal difference learning [15] where the difference of temporally successive predictions is used for adjusting the earlier one. One should be careful, though. If the examples given for learning are fluent all the time, the robustness of the model might start to decrease.

When faced with an unknown state, the best thing to do is often first decrease the uncertainty by for example looking around, and then take action based on what has been revealed. This is called probing. Unfortunately the simple posterior approximation used in this paper does not allow such plans. The future actions (control signals) need to depend on future states but unfortunately they are assumed to be independent here. An interesting continuation is to use another posterior approximation, such as particle filters, for allowing that.

B. Main Results

Selecting actions based on a state-space model instead of based on the observation directly has many benefits: Firstly, it is more resistant to noise because it implicitly involves filtering. Secondly, the observations (without history) do not always carry enough information about the system state. Thirdly, when nonlinear dynamics are modelled by a function approximator such as an multilayer perceptron network, a state-space model can find such a representation of the state that it is more suitable for the approximation and thus more predictable.

When task-oriented identification is used, the state representation becomes such that also the control signals become easier to predict, that is, control becomes easier. The learned policy mapping can also be straightforwardly used for direct control. We think that task-oriented identification should also help indirect control methods but this is yet to be experimentally confirmed.

Nonlinear state-space models seem promising for complex control tasks, where the observations about the system state are incomplete or the dynamics of the system is not well known. The experiments with a simple control task indicated the benefits of the proposed approach. There is still work left in combating high computational complexity and in giving some guarantees or proofs on performance especially in unexpected situations or near boundaries.

Acknowledgement

The authors would like to thank Harri Valpola, Sampsa Laine, Kai Zenger, Heikki Hyötyniemi, and Antti Honkela for fruitful discussions and comments. This research has been funded by the Finnish Centre of Excellence Programme (2000-2005) under the project New Information Processing Principles, and by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

References

[1] B. Anderson and J. Moore. *Optimal Filtering*. Prentice-Hall, Englewood Cliffs, NJ, 1979.

[2] K.J. Åström, P. Albertos, M. Blamke, A. Isidori, W. Schaufelberger, and R. Sanz. *Control of complex systems*. Springer, 2001.

[3] Y. Bar-Shalom. Stochastic dynamic programming: Caution and probing. *IEEE Transactions on Automatic Control*, 26(5):1184–1195, October 1981.

[4] M. J. Beal and Z. Ghahramani. The variational Kalman smoother. Technical Report 003, Gatsby Computational Neuroscience Unit, 2001.

[5] K. Doya. What are the computations in the cerebellum, the basal ganglia, and the cerebral cortex? *Neural Networks*, 12(7):961–974, 1999.

[6] S. Haykin. *Neural Networks – A Comprehensive Foundation, 2nd ed.* Prentice-Hall, 1999.

[7] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[8] H. Kimura and S. Kobayashi. Efficient non-linear control by combining Q-learning with local linear controllers. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 210–219, San Francisco, CA, USA, 1999.

[9] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and B. Likar. Predictive control with Gaussian process models. In *Proceedings of IEEE Region 8 Eurocon 2003: Computer as a Tool*, pages 352–356, 2003.

[10] G.G. Murray and K. Sykes. The variation of hand tremor with force in healthy subjects. *Journal of Physiology*, 191:699–711, 1967.

[11] R. Murray, K. J. Åström, S. P. Boyd, R. W. Brockett, and G. Stein. Future directions in control in an information-rich world. *IEEE Control Systems Magazine*, 23(2):20–33, April 2003.

[12] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[13] M. Nørgaard, O. Ravn, N. K. Poulsen, and L. K. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems*. Springer-Verlag London Limited, 2001.

[14] F. Rosenqvist and A. Karlström. Realisation and estimation of piecewise-linear output-error models. *Automatica*, 41(3):545–551, March 2005.

[15] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[16] S. B. Thrun. The role of exploration in learning control. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, pages 527–559. Van Nostrand Reinhold, Florence, Kentucky, 1992.

[17] H. Valpola and J. Karhunen. An unsupervised ensemble learning method for nonlinear dynamic state-space models. *Neural Computation*, 14(11):2647–2692, 2002.

[18] P. Wawrzynski and A. Pacut. Model-free off-policy reinforcement learning in continuous environment. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1091–1096, Budapest, Hungary, July 2004.

Publication 5

T. Raiko, M. Tornio, A. Honkela, and J. Karhunen. State Inference in Variational Bayesian Nonlinear State-Space Models. In the *Proceedings of the 6th International Conference on Independent Component Analysis and Blind Source Separation (ICA 2006)*, pp. 222–229, Charleston, South Carolina, USA, March 5–8, 2006.

State Inference in Variational Bayesian Nonlinear State-Space Models

Tapani Raiko, Matti Tornio, Antti Honkela, and Juha Karhunen

Helsinki University of Technology,
Neural Networks Research Centre,
P.O. Box 5400, FI-02015 HUT, Espoo, Finland
{tapani.raiko, matti.tornio, antti.honkela, juha.karhunen}@hut.fi

Abstract. Nonlinear source separation can be performed by inferring the state of a nonlinear state-space model. We study and improve the inference algorithm in the variational Bayesian blind source separation model introduced by Valpola and Karhunen in 2002. As comparison methods we use extensions of the Kalman filter that are widely used inference methods in tracking and control theory. The results in stability, speed, and accuracy favour our method especially in difficult inference problems.

1 Introduction

Many applications of source separation methods involve data with some kind of relations between consecutive observations. Examples include relations between neighbouring pixels in images and time series data. Using information on these relations improves the quality of separation results, especially in difficult nonlinear separation problems. Nonlinear modelling of relations may also be useful in linear mixing problems as the dynamics of the time series, for instance, may well be nonlinear.

A method for blind source separation using a nonlinear state-space model is described in [1]. In this paper we study and improve ways of estimating the sources or states in this framework. Efficient solution of the state estimation problem requires taking into account the nonlinear relations between consecutive samples, making it significantly more difficult than source separation in static models. Standard algorithms based on extensions of the Kalman smoother work rather well in general, but may fail to converge when estimating the states over a long gap or when used together with learning the model. We propose solving the problem by improving the variational Bayesian technique proposed in [1] by explicitly using the information on the relation between consecutive samples to speed up convergence.

To tackle just the state estimation (or source separation) part, we will simplify the blind problem by fixing the model weights and other parameters. In [2], linear and nonlinear state-space models are used for blind and semi-blind source separation. Also there the problem is simplified by fixing part of the model.

2 Nonlinear State-Space Models

In nonlinear state-space models, the observation vectors $\mathbf{x}(t)$, $t = 1, 2, \ldots, T$, are assumed to have been generated from unobserved state (or source) vectors $\mathbf{s}(t)$. The model equations are

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t) \tag{1}$$

$$\mathbf{s}(t) = \mathbf{g}(\mathbf{s}(t-1)) + \mathbf{m}(t), \tag{2}$$

Both the mixing mapping $\mathbf{f}$ and the process mapping $\mathbf{g}$ are nonlinear. The noise model for both mixing and dynamical process is often assumed to be Gaussian

$$p(\mathbf{n}(t)) = \mathcal{N}\left[\mathbf{n}(t); \mathbf{0}; \mathbf{\Sigma}_x\right] \tag{3}$$

$$p(\mathbf{m}(t)) = \mathcal{N}\left[\mathbf{m}(t); \mathbf{0}; \mathbf{\Sigma}_s\right], \tag{4}$$

where $\mathbf{\Sigma}_x$ and $\mathbf{\Sigma}_s$ are the noise covariance matrices. In blind source separation, the mappings $\mathbf{f}$ and $\mathbf{g}$ are assumed to be unknown [1] but in this paper we concentrate on the case where they are known.

2.1 Inference Methods

The task of estimating a sequence of sources $\mathbf{s}(1), \ldots, \mathbf{s}(T)$ given a sequence of observations $\mathbf{x}(1), \ldots, \mathbf{x}(T)$ and the model is called inference. In case $\mathbf{f}$ and $\mathbf{g}$ in Eqs. (1) and (2) are linear, the state can be inferred analytically with an algorithm called the *Kalman filter* [3]. In a filter phase, evidence from the past is propagated forward, and in a smoothing phase, evidence from the future is propagated backwards. Only the most recent state can be inferred using the Kalman filter, otherwise the algorithm should be called the *Kalman smoother*. In [4], the Kalman filter is extended for blind source separation from time-varying noisy mixtures.

The idea behind *iterated extended Kalman smoother* [3] (IEKS) is to linearise the mappings $\mathbf{f}$ and $\mathbf{g}$ around the current state estimates using the first terms of the Taylor series expansion. The algorithm alternates between updating the state estimates by Kalman smoothing and renewing the linearisation. When the system is highly nonlinear or the initial estimate is poor, the IEKS may diverge.

The *iterative unscented Kalman smoother* [5,6] (IUKS) replaces the local linearisation of IEKS by a deterministic sampling technique. The sampled points are propagated through the nonlinearities, and a Gaussian distribution is fitted to them. The use of nonlocal information improves convergence and accuracy at the cost of doubling the computational complexity[1]. Still there is no guarantee of convergence.

A recent variant called *backward-smoothing extended Kalman filter* [8] searches for the maximum a posteriori solution to the filtering problem by a guarded Gauss-Newton method. It increases the accuracy further and guarantees convergence at the cost of about hundredfold increase in computational burden.

[1] An even better way of replacing the local linearisation when a multilayer perceptron network is used as a nonlinearity, is described in [7].

Particle filter [9] uses a set of particles or random samples to represent the state distribution. It is a Monte Carlo method developed especially for sequences. The particles are propagated through nonlinearities and there is no need for linearisation nor iterating. Given enough particles, the state estimate approaches the true distribution. Combining the filtering and smoothing directions is not straightforward but there are alternative methods for that. In [10], particle filters are used for non-stationary ICA.

2.2 Variational Bayesian Method

Nonlinear dynamical factor analysis (NDFA) [1] is a variational Bayesian method for learning nonlinear state-space models. The mappings $\mathbf{f}$ and $\mathbf{g}$ in Eqs. (1) and (2) are modelled with multilayer perceptron (MLP) networks whose parameters can be learned from the data. The parameter vector $\boldsymbol{\theta}$ include network weigths, noise levels, and hierarchical priors for them. The posterior distribution over the sources $\mathbf{S} = [\mathbf{s}(1), \dots, \mathbf{s}(T)]$ and the parameters $\boldsymbol{\theta}$ is approximated by a Gaussian distribution $q(\mathbf{S}, \boldsymbol{\theta})$ with some further independency assumptions. Both learning and inference are based on minimising a cost function $\mathcal{C}_{\mathrm{KL}}$

$$\mathcal{C}_{\mathrm{KL}} = \int_{\boldsymbol{\theta}} \int_{\mathbf{S}} q(\mathbf{S}, \boldsymbol{\theta}) \ln \frac{q(\mathbf{S}, \boldsymbol{\theta})}{p(\mathbf{X}, \mathbf{S}, \boldsymbol{\theta})} d\mathbf{S} d\boldsymbol{\theta}, \tag{5}$$

where $p(\mathbf{X}, \mathbf{S}, \boldsymbol{\theta})$ is the joint probability density over the data $\mathbf{X} = [\mathbf{x}(1), \dots, \mathbf{x}(T)]$, sources $\mathbf{S}$, and parameters $\boldsymbol{\theta}$. The cost function is based on Kullback-Leibler divergence between the approximation and the true posterior. It can be split into terms, which helps in studying only a part of the model at a time. The variational approach is less prone to overfitting compared to maximum a posteriori estimates and still fast compared to Monte Carlo methods. See [1] for details.

The variational Bayesian inference algorithm in [1] uses the gradient of the cost function w.r.t. state in a heuristic manner. We propose an algorithm that differs from it in three ways. Firstly, the heuristic updates are replaced by a standard conjugate gradient algorithm [11]. Secondly, the linearisation method from [7] is applied. Thirdly, the gradient is replaced by a vector of approximated total derivatives, as described in the following section.

2.3 Total Derivatives

When updates are done locally, information spreads around slowly because the states of different time slices affect each other only between updates. It is possible to predict this interaction by a suitable approximation. We get a novel update algorithm for the posterior mean of the states by replacing partial derivatives of the cost function w.r.t. state means $\overline{\mathbf{s}}(t)$ by (approximated) total derivatives

$$\frac{d\mathcal{C}_{\mathrm{KL}}}{d\overline{\mathbf{s}}(t)} = \sum_{\tau=1}^{T} \frac{\partial \mathcal{C}_{\mathrm{KL}}}{\partial \overline{\mathbf{s}}(\tau)} \frac{\partial \overline{\mathbf{s}}(\tau)}{\partial \overline{\mathbf{s}}(t)}. \tag{6}$$

They can be computed efficiently using the chain rule and dynamic programming, given that we can approximate the terms $\frac{\partial \overline{\mathbf{s}}(t)}{\partial \overline{\mathbf{s}}(t-1)}$ and $\frac{\partial \overline{\mathbf{s}}(t)}{\partial \overline{\mathbf{s}}(t+1)}$.

Before going into details, let us go through the idea. The posterior distribution of the state $\mathbf{s}(t)$ can be factored into three potentials, one from $\mathbf{s}(t-1)$ (the past), one from $\mathbf{s}(t+1)$ (the future), and one from $\mathbf{x}(t)$ (the observation). We will linearise the nonlinear mappings so that the three potentials become Gaussian. Then also the posterior of $\mathbf{s}(t)$ becomes Gaussian with a mean that is the weighted average of the means of the three potentials, where the weights are the inverse (co)variances of the potentials. A change in the mean of a potential results in a change of the mean of the posterior inversely proportional to their (co)variances.

The terms of the cost function (See Equation (5.6) in [1], although the notation is somewhat different) that relate to $\mathbf{s}(t)$ are

$$\begin{aligned}\mathcal{C}_{\mathrm{KL}}(\mathbf{s}(t)) = &\sum_{i=1}^{m}\left(-\frac{1}{2}\ln \widetilde{s}_{ii}(t) + \frac{1}{2}\Sigma_{sii}^{-1}\left\{\left[\overline{s}_i(t) - \overline{g}_i(\mathbf{s}(t-1))\right]^2 + \widetilde{s}_i(t)\right\}\right)\\ &+ \sum_{j=1}^{m}\frac{1}{2}\Sigma_{sjj}^{-1}\left\{\left[\overline{g}_j(\mathbf{s}(t)) - \overline{s}_j(t+1)\right]^2 + \widetilde{g}_j(\mathbf{s}(t))\right\}\\ &+ \sum_{k=1}^{n}\frac{1}{2}\Sigma_{xkk}^{-1}\left\{\left[\overline{f}_k(\mathbf{s}(t)) - \overline{x}_k(t)\right]^2 + \widetilde{f}_k(\mathbf{s}(t))\right\},\end{aligned} \tag{7}$$

where $\overline{\alpha}$ and $\widetilde{\alpha}$ denote the mean and (co)variance of α over the posterior approximation q respectively and n and m are the dimensionalities of $\mathbf{x}$ and $\mathbf{s}$ respectively. Note that we assume diagonal noise covariances $\mathbf{\Sigma}$. Nonlinearities $\mathbf{f}$ and $\mathbf{g}$ are replaced by the linearisations

$$\widehat{\mathbf{f}}(\mathbf{s}(t)) = \overline{\mathbf{f}}(\mathbf{s}_{\mathrm{cur}}(t)) + \mathbf{J}_f(t)\left[\mathbf{s}(t) - \overline{\mathbf{s}}_{\mathrm{cur}}(t)\right] \tag{8}$$

$$\widehat{\mathbf{g}}(\mathbf{s}(t)) = \overline{\mathbf{g}}(\mathbf{s}_{\mathrm{cur}}(t)) + \mathbf{J}_g(t)\left[\mathbf{s}(t) - \overline{\mathbf{s}}_{\mathrm{cur}}(t)\right], \tag{9}$$

where the subscript cur denotes a current estimate that is constant w.r.t. further changes in $\mathbf{s}(t)$. The minimum of (7) with linearisations can be found at the zero of the gradient:

$$\widetilde{\mathbf{s}}_{\mathrm{opt}}(t) = \left[\mathbf{\Sigma}_s^{-1} + \mathbf{J}_g(t)^{\mathrm{T}}\mathbf{\Sigma}_s^{-1}\mathbf{J}_g(t) + \mathbf{J}_f(t)^{\mathrm{T}}\mathbf{\Sigma}_x^{-1}\mathbf{J}_f(t)\right]^{-1} \tag{10}$$

$$\begin{aligned}\overline{\mathbf{s}}_{\mathrm{opt}}(t) = \widetilde{\mathbf{s}}_{\mathrm{opt}}(t)&\left\{\mathbf{\Sigma}_s^{-1}\left[\overline{\mathbf{g}}(\mathbf{s}_{\mathrm{cur}}(t-1)) + \mathbf{J}_g(t-1)(\overline{\mathbf{s}}(t-1) - \overline{\mathbf{s}}_{\mathrm{cur}}(t-1))\right]\right.\\ &+ \mathbf{J}_g(t)^{\mathrm{T}}\mathbf{\Sigma}_s^{-1}\left[\overline{\mathbf{s}}(t+1) - \overline{\mathbf{g}}(\mathbf{s}_{\mathrm{cur}}(t))\right]\\ &\left.+\mathbf{J}_f(t)^{\mathrm{T}}\mathbf{\Sigma}_x^{-1}\left[\overline{\mathbf{x}}(t) - \overline{\mathbf{f}}(\mathbf{s}_{\mathrm{cur}}(t))\right]\right\}.\end{aligned} \tag{11}$$

The optimum mean reacts to changes in the past and in the future by

$$\frac{\partial \overline{\mathbf{s}}_{\mathrm{opt}}(t)}{\partial \overline{\mathbf{s}}(t-1)} = \widetilde{\mathbf{s}}_{\mathrm{opt}}(t)\mathbf{\Sigma}_s^{-1}\mathbf{J}_g(t-1) \tag{12}$$

$$\frac{\partial \overline{\mathbf{s}}_{\mathrm{opt}}(t)}{\partial \overline{\mathbf{s}}(t+1)} = \widetilde{\mathbf{s}}_{\mathrm{opt}}(t)\mathbf{J}_g(t)^{\mathrm{T}}\mathbf{\Sigma}_s^{-1}. \tag{13}$$

Finally, we assume that the Equations (12) and (13) apply approximately even in the nonlinear case when the subscripts opt are dropped out. The linearisation matrices $\mathbf{J}$ need to be computed anyway [7] so the computational overhead is rather small.

3 Experiments

To experimentally measure the performance of our proposed new method, we used two different data sets. The first data set was generated using a simulated double inverted pendulum system with known dynamics. As the second data set we used real-world speech data with unknown dynamics.

In all the experiments, IEKS and IUKS were run for 50 iterations and NDFA algorithm for 500 iterations. In most cases this was long enough for the algorithms to converge to a local minimum. For comparison purposes, the NDFA experiments were also repeated without using the total derivatives.

Even with a relatively high number of particles, particle smoother performed poorly compared to the iterative algorithms. The results for particle smoother are therefore omitted from the figures. They are however discussed where appropriate. Even though the particle smoother performed relatively poorly, it should be noted that many different schemes exists to improve the performance of particle filters [9], and therefore direct comparison between the iterative algorithms and the plain particle filter algorithm used in these experiments may be somewhat unjustified. The experiments were also repeated with the original NDFA algorithm presented in [1]. The results were quite poor, as was to be excepted, as the heuristic update rules are optimized for learning.

3.1 Double Inverted Pendulum

The double inverted pendulum system [6] (see Figure 1) is a standard benchmark in the field of control. The system consists of a cart and a two-part pole attached to the cart. The system has six states which are cart position on a track, cart velocity, and the angles and the angular velocities of the two attached pendulums. The single control signal is the lateral force applied to the cart. The dynamical equations for the double inverted pendulum system can be found e.g. in [6], in this experiment a discrete system with a time step of $\Delta t = 0.05$ s was simulated using the MATLAB ordinary differential equation solver `ode23`.

To make sure that the learning scheme did not favour the proposed algorithm, standard backpropagation algorithm was used to learn an MLP network to model the system dynamics using a relatively small sample of 2000 input-output pairs. To make this problem more challenging, only the velocity and position of the cart and the angle of the upper pendulum were available as observations, and the rest of the state had to be inferred from these. Experiments were run on ten different data sets with 50 samples each using 5 different initialisations. The final results can be seen in Figure 1.

IEKS suffered from quite serious convergence problems with this data set. These problems were especially bad during the early iterations, but several runs failed to converge to a meaningful result even after the iteration limit was reached. IUKS performed somewhat better, but suffered from some stability problems too. The proposed method was much more robust and did not suffer from stability issues and also performed better on average than the two

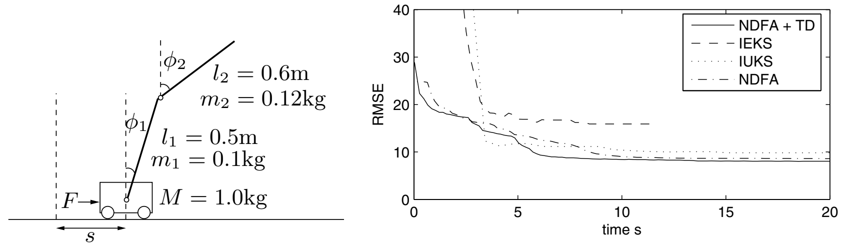


Fig. 1. Inference with the double inverted pendulum system. On the left the schematic of the system, on the right root mean square error plotted against computation time.

Kalman filter based algorithms. It should be noted, however, that in some experiments both IEKS and IUKS converged in only a few iterations, resulting in a superior performance compared to the proposed method. Therefore the problem with IEKS and IUKS may at least partially be related to poor choice of initialisations.

3.2 Speech Spectra

As a real world data set we used speech spectra. The data set consisted of 11200 21 dimensional samples which corresponds to 90 seconds of continuous human speech. The first 10000 samples were used to train a seven dimensional state-space model with the method from [1] and the rest of the data was used in the experiments. This data set poses a somewhat different problem from the double inverted pendulum system. The nonlinearities are not as strong as in the first experiment but the dimensionality of the observation and state spaces are higher, which emphasises the scalability of the methods.

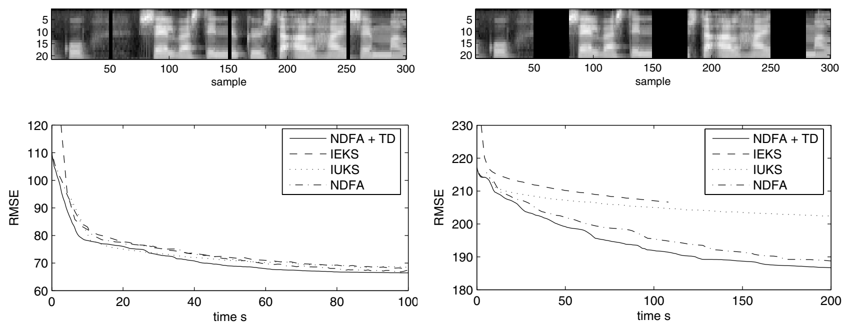


Fig. 2. Inference with the speech data and missing values. On the top one of the data sets used in the experiments (missing values marked in black), on the bottom root mean square error plotted against computation time. Left side figures use a small gap size, right side figures a large gap size.

The test data set was divided into three parts each consisting of 300 samples and all the algorithms were run for each data set with four random initialisations. The final results represent an average over both the different data sets and initialisations.

Since the true state is unknown in this experiment, the mean square error of the reconstruction of missing data was used to compare the different algorithms. Experiments were done with sets of both 3 and 30 consecutive missing samples. The ability to cope with missing values is very important when only partial observations are available or in the case of failures in the observation process. It also has interesting applications in the field of control as reported in [12].

Results can be seen in Figure 2. When missing values are present, especially in the case of the large gap size, the proposed algorithm performs clearly better than the rest of the compared algorithms. Compared to the double inverted pendulum data set, the stability issues with IEKS and IUKS were not as severe, but neither method could cope very well with long gaps of missing values.

4 Discussion and Conclusions

We proposed an algorithm for inference in nonlinear state-space models and compared it to some of the existing methods. The algorithm is based on minimising a variational Bayesian cost function and the novelty is in propagating the gradient through the state sequence. The results were slightly better than any of the comparison methods (IEKS and IUKS). The difference became large in a high-dimensional problem with long gaps in observations.

Our current implementation requires that the nonlinear mappings are modelled as multilayer perceptron networks. Part of the success of our method is due to a linearisation that is specialised to that case [7]. The idea presented in this paper applies in general.

When an algorithm is based on minimising a cost function, it is fairly easy to guarantee convergence. While the Kalman filter is clearly the best choice for inference in linear Gaussian models, the problem with many of the nonlinear generalisation (e.g. IEKS and IUKS) is that they cannot guarantee convergence. Even when the algorithms converge, convergence can be slow. A recent fix for convergence comes with a large computational cost [8] but this paper shows that stable inference can be fast, too.

While this paper concentrates on the case where nonlinear mappings and other model parameters are known, we aim at the case where they should be learned from the data [1]. Blind source separation involves a lot more iterations than the basic source separation. The requirements of a good inference algorithm change, too: There is always the previous estimate of the sources available and most of the time it is already quite accurate.

Acknowledgements

This work was supported in part by the Finnish Centre of Excellence Programme (2000-2005) under the project New Information Processing Principles and by the IST Programme of the European Community under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

References

1. H. Valpola and J. Karhunen, "An unsupervised ensemble learning method for nonlinear dynamic state-space models," *Neural Computation*, vol. 14, no. 11, pp. 2647–2692, 2002.
2. A. Cichocki, L. Zhang, S. Choi, and S.-I. Amari, "Nonlinear dynamic independent component analysis using state-space and neural network models," in *Proc. of the 1st Int. Workshop on Independent Component Analysis and Signal Separation (ICA'99)*, (Aussois, France, January 11-15), pp. 99–104, 1999.
3. B. Anderson and J. Moore, *Optimal Filtering*. Englewood Cliffs, NJ: Prentice-Hall, 1979.
4. V. Koivunen, M. Enescu, and E. Oja, "Adaptive algorithm for blind separation from noisy time-varying mixtures," *Neural Computation*, vol. 13, pp. 2339–2357, 2001.
5. S. Julier and J. Uhlmann, "A new extension of the Kalman filter to nonlinear systems," in *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, 1997.
6. E. A. Wan and R. van der Merwe, "The unscented Kalman filter," in *Kalman Filtering and Neural Networks* (S. Haykin, ed.), pp. 221–280, New York: Wiley, 2001.
7. A. Honkela and H. Valpola, "Unsupervised variational Bayesian learning of nonlinear models," in *Advances in Neural Information Processing Systems 17* (L. Saul, Y. Weiss, and L. Bottou, eds.), pp. 593–600, Cambridge, MA, USA: MIT Press, 2005.
8. M. Psiaki, "Backward-smoothing extended Kalman filter," *Journal of Guidance, Control, and Dynamics*, vol. 28, Sep–Oct 2005.
9. A. Doucet, N. de Freitas, and N. J. Gordon, *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.
10. R. Everson and S. Roberts, "Particle filters for non-stationary ICA," in *Advances in Independent Component Analysis* (M. Girolami, ed.), pp. 23–41, Springer-Verlag, 2000.
11. R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The Computer Journal*, vol. 7, pp. 149–154, 1964.
12. T. Raiko and M. Tornio, "Learning nonlinear state-space models for control," in *Proc. Int. Joint Conf. on Neural Networks (IJCNN'05)*, (Montreal, Canada), pp. 815–820, 2005.

Publication 6

T. Raiko. Nonlinear Relational Markov Networks with an Application to the Game of Go. In the *Proceedings of the International Conference on Artificial Neural Networks (ICANN 2005)*, pp. 989–996, Warsaw, Poland, September 11–15, 2005.

6

Nonlinear Relational Markov Networks with an Application to the Game of Go

Tapani Raiko

Neural Networks Research Centre, Helsinki University of Technology,
P.O.Box 5400, FI-02015 HUT, Espoo, FINLAND
Tapani.Raiko@hut.fi

Abstract. It would be useful to have a joint probabilistic model for a general relational database. Objects in a database can be related to each other by indices and they are described by a number of discrete and continuous attributes. Many models have been developed for relational discrete data, and for data with nonlinear dependencies between continuous values. This paper combines two of these methods, relational Markov networks and hierarchical nonlinear factor analysis, resulting in joining nonlinear models in a structure determined by the relations in the data. The experiments on collective regression in the board game go suggest that regression accuracy can be improved by taking into account both relations and nonlinearities.

1 Introduction

Growing amount of data is collected every day in all fields of life. For the purpose of automatic analysis, prediction, denoising, classification etc. of data, a huge number of models have been created. It is natural that a specific model for a specific purpose works often the best, but still, a general method to handle any kind of data would be very useful. For instance, if an artificial brain has a large number of completely separate modules for different tasks, the interaction between the modules becomes difficult. Probabilistic modelling provides a well-grounded framework for data analysis. This paper describes a probabilistic model that can handle data with relations as well as discrete and continuous values with nonlinear dependencies.

Terminology: Using Prolog notation, we write $\mathrm{knows}(\mathrm{alex}, \mathrm{bob})$ for stating a fact that the knows relation holds between the objects alex and bob, that is, Alex knows Bob. The arity of the relation tells how many objects are involved. The knows relation is binary, that is, between two objects, but in general relations can be of any arity. The atom $\mathrm{knows}(\mathrm{alex}, B)$ matches all the instances where the variable B represents an object known by Alex. In this paper, the terms are restricted to constants and variables, that is, compound terms such as $\mathrm{thinks}(A, \mathrm{knows}(B, A))$ are not considered. For every relation that is logically true, there are associated attributes $\mathbf{x}$, say a class label or a vector of real numbers. The attributes $\mathbf{x}(\mathrm{knows}(A, B))$ describe how well A knows B and whether A likes or dislikes B. The attribute vector $\mathbf{x}(\mathrm{con}(A))$ describes what kind of a consumer the person A is. Given a relational database describing relationships between people and their consuming habits, we might study the dependencies that might be found. For instance, some people cloth like their idols, and nonsmokers tend to be

KNOWS

who	whom	how
Alex	Bob	friend
Bob	Carl	neighbour
Carl	Alex	colleague

CONSUMER

who	smoker	...
Alex	no	...
Bob	no	...
Carl	no	...

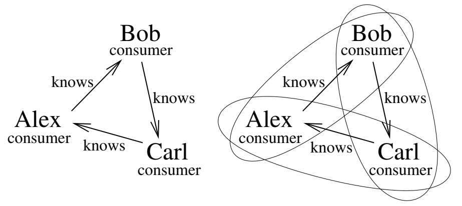


Fig. 1. Consider a relational database describing the relationships and consumer habits of three people. The two tables are shown on the left. On the right, the database is represented graphically, with the occurrences of the template $(\mathrm{con}(A), \mathrm{knows}(A, B), \mathrm{con}(B))$ marked with ovals on the very right.

friends with nonsmokers. The modelling can be done for instance by finding all occurrences of the template $(\mathrm{con}(A), \mathrm{knows}(A, B), \mathrm{con}(B))$ in the data and studying the distribution of the corresponding attributes. The situation is depicted in Figure 1.

Bayesian networks[6] are popular statistical models based on a directed graph. The graph has to be acyclic, which is in line with the idea that the arrows represent causality: an occurrence cannot be its own cause. In relational generalisations of Bayesian networks [7], the graphical structure is determined by the data. Often it can be assumed that the data does not contain cycles, for instance in the case when the direction of the arrows is always from the past to the future. Sometimes the data has cycles, like in Figure 1. Markov networks [6], on the other hand, are based on undirected graphical models. A Markov network does not care whether A caused B or vice versa, it is interested only whether there is a dependency or not.

2 Model Description

This section describes the models that are combined into nonlinear relational Markov networks.

2.1 Hierarchical Nonlinear Factor Analysis (HNFA)

In (linear) factor analysis, continuous valued observation vectors $\mathbf{x}(t)$ are generated from unknown factors (or sources) $\mathbf{s}(t)$, a bias vector $\mathbf{b}$, and noise $\mathbf{n}(t)$ by $\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{b} + \mathbf{n}(t)$. The factors and noise are assumed to be Gaussian and independent. The index t may represent time or the object of the observation. The mapping $\mathbf{A}$, the factors, and parameters such as the noise variances are found using Bayesian learning. Factor analysis is close to principal component analysis (PCA). The unknown factors may represent some real phenomena, or they may just be auxillary variables for inducing a dependency between the observations.

Hierarchical nonlinear factor analysis (HNFA) [11] generalises factor analysis by adding more layers of factors that form a multi-layer perceptron type of a network. In this paper, there are two layers of factors $\mathbf{h}$ and $\mathbf{s}$, and the mappings are:

$$\mathbf{h}(t) = \mathbf{B}\mathbf{s}(t) + \mathbf{b} + \mathbf{n}_h(t) \tag{1}$$

$$\mathbf{x}(t) = \mathbf{A}\mathbf{f}[\mathbf{h}(t)] + \mathbf{C}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}_x(t)\,, \tag{2}$$

where the nonlinearity $\mathbf{f}(\xi) = \exp(-\xi^2)$ operates on each element separately. HNFA can easily be implemented using the Bayes Blocks software library [10, 12]. The update rules are automatically derived in a manner shortly described below.

The unknown variables $\boldsymbol{\theta}$ (factors, mappings, and the parameters) are learned from data with variational Bayesian learning [4]. A parametric distribution $q(\boldsymbol{\theta})$ over the unknown variables $\boldsymbol{\theta}$ is fitted to the true posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{X})$ where the matrix $\boldsymbol{X}$ contains all the observations $\mathbf{x}(t)$. The misfit is measured by Kullback-Leibler divergence $D(\cdot \parallel \cdot)$. An additional term $-\log p(\boldsymbol{X})$ is included to avoid calculation of the model evidence term $p(\boldsymbol{X}) = \int p(\boldsymbol{X}, \boldsymbol{\theta}) d\boldsymbol{\theta}$. The cost function is

$$\mathcal{C} = D(q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta}|\boldsymbol{X})) - \log p(\boldsymbol{X}) = \left\langle \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{X}, \boldsymbol{\theta})} \right\rangle , \tag{3}$$

where $\langle \cdot \rangle$ denotes the expectation over distribution $q(\boldsymbol{\theta})$. Note that since $D(q \parallel p) \geq 0$, it follows that the cost function provides a lower bound for the model evidence $p(\boldsymbol{X}) \geq \exp(-\mathcal{C})$. The posterior approximation $q(\boldsymbol{\theta})$ is chosen to be Gaussian with a diagonal covariance matrix.

It is possible, though slightly impractical, to model also discrete values in HNFA by using the discrete variable with a soft-max prior [12]. In the binary case, the ith component of $\mathbf{x}(t)$ is left as a latent auxiliary variable, and an observed binary variable $y(t)$ is conditioned by $p(y(t) = 1 \mid x_i(t)) = \frac{\exp x_i(t)}{1+\exp x_i(t)}$. The general discrete case follows analogously requiring more than one auxiliary component of $\mathbf{x}(t)$. The experiments in Section 3 use a thousand copies of a binary variable having the same conditional probability. They can be united into one variable by multiplying its cost by one thousand. Observing 800 ones and 200 zeros corresponds to fixing the variable to a distribution of 0.8 times one and 0.2 times zero.

2.2 Relational Markov Networks (RMN)

A relational Markov network (RMN) [9] is a model for data with relations and discrete attributes. It is specified by a set of clique templates $\mathbf{C}$ and corresponding potentials $\boldsymbol{\Phi}$. Using the example in the introduction, a model can be formed by defining a single clique template $C = (\mathrm{con}(A), \mathrm{knows}(A, B), \mathrm{con}(B))$ and the corresponding potential ϕ_C over $\mathbf{x}(C)$ which is (a subset of) the concatenation of attribute vectors $\mathbf{x}(\mathrm{con}(A))$, $\mathbf{x}(\mathrm{knows}(A, B))$, and $\mathbf{x}(\mathrm{con}(B))$. Given a relational database, the RMN produces an unrolled Markov network over all the attributes $\boldsymbol{X}$. The cliques $c \in C(\mathcal{I})$ instantiated by a template C share the same clique potential ϕ_C. The combined probabilistic model is $p(\boldsymbol{X}) = \frac{1}{Z} \prod_{C \in \mathbf{C}} \prod_{c \in C(\mathcal{I})} \phi_C(\mathbf{x}(c))$, where Z is a normalisation constant and $C(\mathcal{I})$ contains all the instantiations of the template C. In general, a template can be any boolean formula over the relations.

The general inference task is to compute the posterior distribution over all the variables $\boldsymbol{X}$. The network induced by data can be very large and densely connected, so exact inference is often intractable [9]. The belief propagation (BP) algorithm [6] is guaranteed to converge to the correct marginal probabilities only for singly connected Markov networks, but it is used as a good approximation also in the loopy case. The learning task, or the estimation of the potentials $\boldsymbol{\Phi}$ is done using the maximum a posteriori criterion. It requires an iterative algorithm alternating between updating the parameters of the potentials and running the inference algorithm on the unrolled Markov network.

2.3 Nonlinear Relational Markov Networks (NRMN)

In nonlinear relational Markov networks (NRMN), the clique potentials are replaced by a probability density function for continuous values, in this case HNFA[1] The combination of these two methods is not completely straightforward. For instance, marginalisation required by the BP algorithm is often difficult with nonlinear models. Also, algorithmic complexity needs to be considered, since the model will be quite demanding. One of the key points is to use probability densities p in place of potentials $\boldsymbol{\Phi}$. Then, overlapping templates give multiple probability functions for some variable and they are combined using the product-of-experts combination rule described below.

Combination Rules: One of the non-trivialities in making relational extensions of probabilistic models is the so called combination rule [7]. When the structure of the graphical model is determined by the data, one cannot know in advance how many links there are for each node. One solution is to use combination rules such as the noisy-or. Combination rule transforms a number of probability functions into one. Noisy-or does not generalise well to continuous values, but two alternatives are introduced below.

Using a Markov network and the BP algorithm corresponds to using probability densities as potentials and the *maximum entropy* combination rule. The probability densities $p_C(\mathbf{x}(C))$ are combined to form the joint probability distribution by maximising the entropy of $p(\boldsymbol{X})$ given that all instantiations of $p_C(\mathbf{x}(C))$ coalesce with the corresponding marginals of $p(\boldsymbol{X})$. For singly connected networks, this means that the joint distribution is $p(\boldsymbol{X}) = \prod_c p_c(\mathbf{x}(c)) / \prod_k p_k(\mathbf{x}(k))$, where k runs over pairs of instantiations of templates and $\mathbf{x}(k)$ contains the shared attributes in those pairs. Marginalisation of nonlinear models cannot usually be done exactly and therefore one should be very careful with the denominator. Also, one should take care in handling loops.

In the *product-of-experts* (PoE) combination rule, the logarithm of the probability density of each variable is the average of the logarithms of the probability functions that the variable is included in: $p(x) \propto \sqrt[n]{\prod_{C \in \mathbf{C}} \prod_{c \in C(\mathcal{I})} p_C(x)}$ for all $x \in \boldsymbol{X}$, where only those n instantiated templates c that contain x, are considered. PoE is easy to implement in the variational Bayesian framework because the term in the cost function (3) can be split into familiar looking terms. Consider the combination of two probability functions $p_1(x)$ and $p_2(x)$ (that are assumed to be independent):

$$\left\langle \log \frac{q(x)}{\sqrt{p_1(x)p_2(x)}} \right\rangle = \frac{\left\langle \log \frac{q(x)}{p_1(x)} \right\rangle + \left\langle \log \frac{q(x)}{p_2(x)} \right\rangle}{2}. \tag{4}$$

A characteristic of PoE is that implicit weighting happens in some sense automatically. When one of the experts gives a distribution with a large variance and another one with a small variance, the combination is close to the one with small variance.

Inference in Loopy Networks: Inferring unobserved attributes in a database is in this case an iterative process which should end up in a cohesive whole. Information can traverse through multiple relations. [2] The basic element in the inference algorithm

[1] One could also think in terms of e.g. a mixture model.

[2] In mixture of experts (MoE), it is enough when only one of the experts explains the data even if all the other disagree. The ignored experts will not pass information on. This explains why the author did not consider MoE as a combination rule.

of Bayes Blocks is the update of the posterior approximation $q(\cdot)$ of a single unknown variable, assuming the rest of the distribution fixed. The update is done such that the cost function (3) is minimised. One should note that when the distribution over the Markov blanket of a variable is fixed, the local update rules apply, regardless of any loops in the network. Therefore the use of local update rules is well founded, that is, local inference in a loopy network does not bring any additional heuristicity to the system. Also, since the inference is based on minimising a cost function, the convergence is guaranteed, unlike in the BP algorithm.

Learning: The learning or parameter estimation problem is to find the probability functions associated with the given clique templates $\mathbf{C}$. Now that we use probability functions instead of potentials, it is possible in some cases to separate the learning problem into parts. For each template $C \in \mathbf{C}$, find the appropriate instantiations $c \in C(\mathcal{I})$ and collect the associated attributes $\mathbf{x}(c)$ into a table. Learn a HNFA model for this table ignoring the underlying relations. This divide-and-conquer strategy makes learning comparatively fast, because all the interaction is avoided. There are some cases that forbid this. If the data contains missing values, they need to be inferred using the method in the previous paragraph. Also, it is possible to train experts cooperatively rather than separately [3].

Clique Templates: In data mining, so called frequent sets are often mined from binary data. Frequent sets are groups of binary variables that get the value 1 together often enough to be called frequent. The generalisation of this concept to continuous values could be called the *interesting sets*. An interesting set contains variables that have such strong mutual dependencies that the whole is considered interesting. The methodology of inductive logic programming could be applied to finding interesting clique templates. The definition of a measure for interestingness is left as future work. Note that the divide-and-conquer strategy described in the previous paragraph becomes even more important if one needs to consider different sets of templates. One can either learn a model for each template separately and then try combinations with the learned models, or try a combination of templates and learn cooperatively the models for them. Naturally the number of templates is much smaller than the number of combinations and thus the first option is computationally much cheaper.

So what are meaningful candidates for clique templates? For instance, the template $(\mathrm{con}(A), \mathrm{con}(B))$ does not make much sense. Variables A and B are not related to each other, so when all pairs are considered, $\mathrm{con}(A)$ and $\mathrm{con}(B)$ are always independent and thus uninteresting by definition. In general, a template is uninteresting, if it can be split into two parts that do not share any variables. When considering large templates, the number of involved attributes grows large as well, which makes learning more involved. An interesting possibility is to make a hierarchical model. When a large template contains others as subtemplates, one can use the factors $\mathbf{s}$ in Eq. (1, 2) of the subtemplate as the attributes for the large template. The factors already capture the internal structure of the subtemplates and thus the probabilistic model of the large template needs only to concentrate on the structure between its subtemplates.

3 Experiments with the Game of Go

Game of Go: Go is an ancient oriental board game. Two players, black and white, alternately place stones on the empty points of the board until they both pass. The standard board is 19 by 19 (i.e. the board has 19 lines by 19 lines), but 13-by-13 and 9-by-9 boards can also be used. The game starts with an empty board and ends when it is divided into black and white areas. The one who has the larger area wins. Stones of one colour form a block when they are 4-connected. Empty points that are 4-connected to a block are called its liberties. When a block loses its last liberty, it is removed from the board. After each move, surrounded opponent blocks are removed and only after that, it is checked whether the block of the played move has liberties or not. There are different rulesets that define more carefully what a "larger area" is, whether suicide is legal or not, and how infinite repetitions are forbidden.

Computer Go: Of all games of skill, go is second only to chess in terms of research and programming efforts spent. While go programs have advanced considerably in the last 10-15 years, they can still be beaten easily by human players of moderate skill. [5] One of the reasons behind the difficulty of static board evaluation is the fact that there are stones on board that will eventually be captured, but not in near future. In many cases experienced go players can classify these dead stones with ease, but using a simple look-ahead to determine the status of stones is not always feasible since it might take dozens of moves to actually capture the stones.

Experiment Setting: The goal of the experiments was to learn to determine the status of the stones without any lookahead. An example situation is given in Figure 2. The data was generated using a go-playing program called Go81 [8] set on level 1 and using randomness to have variability. By playing the game from the current position to the end a thousand times, one gets an estimate of who is going to own each point on the board. Information on the board states was saved to a relational database with two tables for learning an NRMN. The $\mathbf{x}(\mathrm{block}(A))$ contains the colour, the number of liberties, the size, distances from the edges, influence features in the spirit of [1], and finally the count of how many times the block survives in the 1000 possible futures. The $\mathrm{ally}(A, B)$ and $\mathrm{enemy}(A, B)$ contain a measure of strength of the connection between the blocks A and B estimated using similar influence features [1]. Only the pairs with a strong enough influence on each other (> 0.02) were included. One thousand 13-by-13 board positions after playing 2 to 60 moves were used for learning.

Two clique templates, $((\mathrm{block}(A), \mathrm{ally}(A, B), \mathrm{block}(B))$ and an analogous one for enemy, were used. HNFA models were taught with 28 attributes of the two blocks and the pair. The dimensionality of the $\mathbf{s}$ layer was 8. The learning algorithm pruned the dimensionality of the $\mathbf{h}$ layer to 41 for allies and to 47 for enemies. The models were learned for 500 sweeps through the data. A linear factor analysis model was learned with the same data for comparison. A separate collection of 81 board positions with 1576 blocks was used for testing. The status of each block was now hidden from the model and only the other attributes were known. With inference in the network, the status were collectively regressed. As a comparison experiment, the inferences were also done separately, and combined only in the end. Inference required from four to thirty iterations to converge. As a postprocessing step, the regressed survival probabilities $\hat{x}$

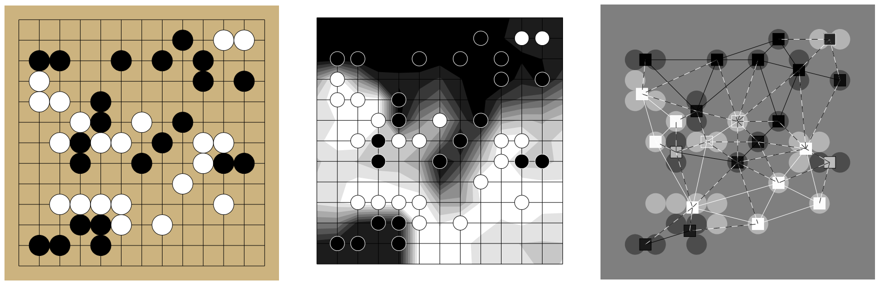


Fig. 2. The leftmost subfigure shows the board of a go game in progress. In the middle, the expected owner of each point is visualised with the shade of grey. For instance, the two white stones in the upper right corner are very likely to be captured. The rightmost subfigure shows the blocks with their expected owner as the colour of the square. Pairs of related blocks are connected with a line which is dashed when the blocks are of opposing colours.

were modified with a simple three-parameter function $\hat{x}_{new} = a\hat{x}^b + c$ and the three parameters that gave the smallest error for each setting, were used.

Results: The table below shows the root mean square (rms) errors for inferring the survival probabilities of the blocks in test cases. They can be compared to the standard deviation 0.2541 of the probabilities.

rms error	Linear	Nonlinear
Separate regression	0.2172	0.2052
Collective regression	0.2171	0.2037

As expected, nonlinear models were better than linear ones and collective regression was better than separate regression.

4 Discussion and Conclusion

A traditional Markov network was applied for statically determining the status of the go board in [2]. Games played by people were used as data. Humans play the game better, but still, this approach has an important downside. The data contains only one possible future for each board position whereas a computer player can produce many possible futures. At the learning stage, all those futures can be used together for the computational price of one. Also, stones that are provably determined to be captured under optimal play (*dead*), might still be useful: By threathening to revive them, the player can gain elsewhere. When data is gathered with unoptimal play, the stones are marked as *not quite dead*, which might be desirable.

NRMN includes a probabilistic model only for the attributes and not for the logical relations. Link uncertainty means that one models the possibility of a certain relation to exist or not. Actually one can model link uncertainty using just the proposed methodology. All the uncertain relations are assumed to be logically true and an additional binary attribute is included to mark whether the link exists or not. One only needs to take into account that when this binary attribute gets the value zero, the dependencies between the other attributes are not modelled. Also, time series data can be represented using relations $\mathrm{obs}(T)$ for the observations at time T and $\mathrm{ensues}(T1, T2)$ to denote that the time indices $T1$ and $T2$ are adjacent. These two examples give light to the generality of the proposed method. In [12], HNFA is augmented with a variance model. Modelling variances would be important also in the NRMN setting, because then each expert would produce an estimate of its accuracy and thus implicitly a weight compared to other experts. In [9], relational Markov networks were constructed to be discriminative so that the model is specialised to classification. The same could be applied here.

Conclusion: A model was proposed for data containing both relations and nonlinear dependencies. The model was built by combining two state-of-the art probabilistic models, hierarchical nonlinear factor analysis and relational Markov networks by using the product-of-experts combination rule. Many simplifying assumptions were made, such as diagonality of the posterior covariance matrix, and separate learning of experts. Also, learning the model structure (the set of clique templates) was left as future work. Experiments with the game of go give promise for the proposed methodology.

Acknowledgements: The author thanks Kristian Kersting, Harri Valpola, Markus Harva, and Alexander Ilin for useful discussions and comments. This research has been funded by the Finnish Centre of Excellence Programme (2000-2005) under the project New Information Processing Principles, and by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the author's views.

References

1. B. Bouzy. Mathematical morphology applied to computer go. *IJPRAI*, 17(2), 2003.
2. T. Graepel D. Stern and D. MacKay. Modelling uncertainty in the game of Go. In *Proc. of the Conference on Neural Information Processing Systems*, Vancouver, December 2004.
3. G.E. Hinton. Modelling high-dimensional data by combining simple experts. In *Proc. AAAI-2000*, Austin, Texas.
4. M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. In M. Jordan, editor, *Learning in Graphical Models*, pages 105–161. The MIT Press, Cambridge, MA, USA, 1999.
5. M. Müller. Computer Go. *Special issue on games of Artificial Intelligence Journal*, 2001.
6. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
7. L. De Raedt and K. Kersting. Probabilistic logic learning. *ACM-SIGKDD Explorations, special issue on Multi-Relational Data Mining*, 5(1):31–48, July 2003.
8. T. Raiko. The go-playing program called Go81. In *Proceedings of the Finnish Artificial Intelligence Conference, STeP 2004*, pages 197–206, Helsinki, Finland, 2004.
9. B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proc. Conference on Uncertainty in Artificial Intelligence (UAI02)*, Edmonton, 2002.
10. H. Valpola, A. Honkela, M. Harva, A. Ilin, T. Raiko, and T. Östman. Bayes blocks software library. `http://www.cis.hut.fi/projects/bayes/software/`, 2003.
11. H. Valpola, T. Östman, and J. Karhunen. Nonlinear independent factor analysis by hierarchical models. In *Proc. ICA2003*, pages 257–262, Nara, Japan, 2003.
12. H. Valpola, T. Raiko, and J. Karhunen. Building blocks for hierarchical latent variable models. In *Proc. ICA2001*, pages 710–715, San Diego, USA, 2001.

Publication 7

K. Kersting, L. De Raedt, and T. Raiko. Logical Hidden Markov Models. In the *Journal of Artificial Intelligence Research*, Volume 25, pp. 425–456, April, 2006.

Journal of Artificial Intelligence Research 25 (2006) 425–456 Submitted 12/04; published 4/06

Logical Hidden Markov Models

Kristian Kersting KERSTING@INFORMATIK.UNI-FREIBURG.DE
Luc De Raedt DERAEDT@INFORMATIK.UNI-FREIBURG.DE
Institute for Computer Science
Albert-Ludwigs-Universität Freiburg
Georges-Koehler-Allee 079
D-79110 Freiburg, Germany

Tapani Raiko TAPANI.RAIKO@HUT.FI
Laboratory of Computer and Information Science
Helsinki University of Technology
P.O. Box 5400
FIN-02015 HUT, Finland

Abstract

Logical hidden Markov models (LOHMMs) upgrade traditional hidden Markov models to deal with sequences of structured symbols in the form of logical atoms, rather than flat characters.

This note formally introduces LOHMMs and presents solutions to the three central inference problems for LOHMMs: evaluation, most likely hidden state sequence and parameter estimation. The resulting representation and algorithms are experimentally evaluated on problems from the domain of bioinformatics.

1. Introduction

Hidden Markov models (HMMs) (Rabiner & Juang, 1986) are extremely popular for analyzing sequential data. Application areas include computational biology, user modelling, speech recognition, empirical natural language processing, and robotics. Despite their successes, HMMs have a major weakness: they handle only sequences of flat, i.e., unstructured symbols. Yet, in many applications the symbols occurring in sequences are structured. Consider, e.g., sequences of UNIX commands, which may have parameters such as `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, . . .Thus, commands are essentially structured. Tasks that have been considered for UNIX command sequences include the prediction of the next command in the sequence (Davison & Hirsh, 1998), the classification of a command sequence in a user category (Korvemaker & Greiner, 2000; Jacobs & Blockeel, 2001), and anomaly detection (Lane, 1999). Traditional HMMs cannot easily deal with this type of structured sequences. Indeed, applying HMMs requires either 1) ignoring the structure of the commands (i.e., the parameters), or 2) taking all possible parameters explicitly into account. The former approach results in a serious information loss; the latter leads to a combinatorial explosion in the number of symbols and parameters of the HMM and as a consequence inhibits generalization.

The above sketched problem with HMMs is akin to the problem of dealing with structured examples in traditional machine learning algorithms as studied in the fields of inductive logic programming (Muggleton & De Raedt, 1994) and multi-relational learning (Džeroski & Lavrač, 2001). In this paper, we propose an (inductive) logic programming framework, Logical HMMs (LOHMMs), that upgrades HMMs to deal with structure. The key idea underlying LOHMMs is to employ logical atoms as structured (output and state) symbols. Using logical atoms, the above UNIX command sequence can be represented as $\mathtt{emacs(lohmms.tex)}, \mathtt{ls}, \mathtt{latex(lohmms.tex)}, \ldots$ There are two important motivations for using logical atoms at the symbol level. First, *variables* in the atoms allow one to make abstraction of specific symbols. E.g., the logical atom $\mathtt{emacs(X, tex)}$ represents all files $\mathtt{X}$ that a LaTeX user $\mathtt{tex}$ could edit using $\mathtt{emacs}$. Second, *unification* allows one to share information among states. E.g., the sequence $\mathtt{emacs(X, tex)}, \mathtt{latex(X, tex)}$ denotes that the same file is used as an argument for both Emacs and LaTeX.

The paper is organized as follows. After reviewing the logical preliminaries, we introduce LOHMMs and define their semantics in Section 3; in Section 4, we upgrade the basic HMM inference algorithms for use in LOHMMs; we investigate the benefits of LOHMMs in Section 5: we show that LOHMMs are strictly more expressive than HMMs, that they can be — by design — an order of magnitude smaller than their corresponding propositional instantiations, and that unification can yield models, which better fit the data. In Section 6, we empirically investigate the benefits of LOHMMs on real world data. Before concluding, we discuss related work in Section 7. Proofs of all theorems can be found in the Appendix.

2. Logical Preliminaries

A *first-order alphabet* Σ is a set of relation symbols $\mathtt{r}$ with arity $m \geq 0$, written $\mathtt{r}/m$, and a set of functor symbols $\mathtt{f}$ with arity $n \geq 0$, written $\mathtt{f}/n$. If $n = 0$ then $\mathtt{f}$ is called a constant, if $m = 0$ then $\mathtt{p}$ is called a propositional variable. (We assume that at least one constant is given.) An *atom* $\mathtt{r}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$ is a relation symbol $\mathtt{r}$ followed by a bracketed n-tuple of terms $\mathtt{t}_i$. A *term* $\mathtt{t}$ is a variable $\mathtt{V}$ or a functor symbol $\mathtt{f}(\mathtt{t}_1, \ldots, \mathtt{t}_k)$ immediately followed by a bracketed k-tuple of terms $\mathtt{t}_i$. Variables will be written in upper-case, and constant, functor and predicate symbols lower-case. The symbol _ will denote anonymous variables which are read and treated as distinct, new variables each time they are encountered. An *iterative clause* is a formula of the form $\mathtt{H} \leftarrow \mathtt{B}$ where $\mathtt{H}$ (called *head*) and $\mathtt{B}$ (called *body*) are logical atoms. A substitution $\theta = \{\mathtt{V}_1/\mathtt{t}_1, \ldots, \mathtt{V}_n/\mathtt{t}_n\}$, e.g. $\{\mathtt{X/tex}\}$, is an assignment of terms $\mathtt{t}_i$ to variables $\mathtt{V}_i$. Applying a substitution σ to a term, atom or clause $\mathtt{e}$ yields the instantiated term, atom, or clause $\mathtt{e}\sigma$ where all occurrences of the variables $\mathtt{V}_i$ are simultaneously replaced by the term $\mathtt{t}_i$, e.g. $\mathtt{ls(X)} \leftarrow \mathtt{emacs(F, X)}\{\mathtt{X/tex}\}$ yields $\mathtt{ls(tex)} \leftarrow \mathtt{emacs(F, tex)}$. A substitution σ is called a *unifier* for a finite set S of atoms if $S\sigma$ is singleton. A unifier θ for S is called a *most general unifier* (MGU) for S if, for each unifier σ of S, there exists a substitution γ such that $\sigma = \theta\gamma$. A term, atom or clause $\mathtt{E}$ is called *ground* when it contains no variables, i.e., $vars(\mathtt{E}) = \emptyset$. The *Herbrand base* of Σ, denoted as hb_Σ, is the set of all ground atoms constructed with the predicate and functor symbols in Σ. The set $G_\Sigma(\mathtt{A})$ of an atom $\mathtt{A}$ consists of all ground atoms $\mathtt{A}\theta$ that belong to hb_Σ.

3. Logical Hidden Markov Models

The logical component of a traditional HMM corresponds to a *Mealy machine* (Hopcroft & Ullman, 1979), i.e., a finite state machine where the output symbols are associated with transitions. This is essentially a propositional representation because the symbols used to represent states and output symbols are flat, i.e. not structured. The key idea underlying LOHMMs is to replace these flat symbols by abstract symbols. An abstract symbol $\mathtt{A}$ is — by definition — a logical atom. It is abstract in that it represents the set of all ground, i.e., variable-free atoms of $\mathtt{A}$ over the alphabet Σ, denoted by $G_\Sigma(\mathtt{A})$. Ground atoms then play the role of the traditional symbols used in a HMMs.

Example 1 *Consider the alphabet Σ_1 which has as constant symbols* $\mathtt{tex}$*,* $\mathtt{dvi}$*,* $\mathtt{hmm1}$*, and* $\mathtt{lohmm1}$*, and as relation symbols* $\mathtt{emacs/2}$*,* $\mathtt{ls/1}$*,* $\mathtt{xdvi/1}$*,* $\mathtt{latex/2}$*. Then the atom* $\mathtt{emacs(File, tex)}$ *represents the set* $\{\mathtt{emacs(hmm1, tex)}, \mathtt{emacs(lohmm1, tex)}\}$*. We assume that the alphabet is typed to avoid useless instantiations such as* $\mathtt{emacs(tex, tex)}$*).*

The use of atoms instead of flat symbols allows us to analyze logical and structured sequences such as $\mathtt{emacs(hmm1, tex)}, \mathtt{latex(hmm1, tex)}, \mathtt{xdvi(hmm1, dvi)}$.

Definition 1 Abstract transition *are expressions of the form* $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ *where* $p \in [0, 1]$*, and* $\mathtt{H}$*,* $\mathtt{B}$ *and* $\mathtt{O}$ *are atoms. All variables are implicitly assumed to be universally quantified, i.e., the scope of variables is a single abstract transition.*

The atoms $\mathtt{H}$ and $\mathtt{B}$ represent abstract states and $\mathtt{O}$ represents an abstract output symbol. The semantics of an abstract transition $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ is that if one is in one of the states in $G_\Sigma(\mathtt{B})$, say $\mathtt{B}\theta_\mathtt{B}$, one will go with probability p to one of the states in $G_\Sigma(\mathtt{H}\theta_\mathtt{B})$, say $\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H}$, while emitting a symbol in $G_\Sigma(\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H})$, say $\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}\theta_\mathtt{O}$.

Example 2 *Consider* $c \equiv 0.8 : \mathtt{xdvi(File, dvi)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, tex)}$*. In general* $\mathtt{H}$*,* $\mathtt{B}$ *and* $\mathtt{O}$ *do not have to share the same predicate. This is only due to the nature of our running example. Assume now that we are in state* $\mathtt{latex(hmm1, tex)}$*, i.e.* $\theta_\mathtt{B} = \{\mathtt{File/hmm1}\}$*. Then c specifies that there is a probability of 0.8 that the next state will be in* $G_{\Sigma_1}(\mathtt{xdvi(hmm1, dvi)}) = \{\mathtt{xdvi(hmm1, dvi)}\}$ *(i.e., the probability is* 0.8 *that the next state will be* $\mathtt{xdvi(hmm1, dvi)}$*), and that one of the symbols in* $G_{\Sigma_1}(\mathtt{latex(hmm1)}) = \{\mathtt{latex(hmm1)}\}$ *(i.e.,* $\mathtt{latex(hmm1)}$*) will be emitted. Abstract states might also be more complex such as* $\mathtt{latex(file(FileStem, FileExtension), User)}$

The above example was simple because $\theta_\mathtt{H}$ and $\theta_\mathtt{O}$ were both empty. The situation becomes more complicated when these substitutions are not empty. Then, the resulting state and output symbol sets are not necessarily singletons. Indeed, for the transition $0.8 : \mathtt{emacs(File', dvi)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, tex)}$ the resulting state set would be $G_{\Sigma_1}(\mathtt{emacs(File', dvi)}) = \{\mathtt{emacs(hmm1, tex)}, \mathtt{emacs(lohmm1, tex)}\}$. Thus the transition is non-deterministic because there are two possible resulting states. We therefore need a mechanism to assign probabilities to these possible alternatives.

Definition 2 *The selection distribution μ specifies for each abstract state and observation symbol* $\mathtt{A}$ *over the alphabet Σ a distribution* $\mu(\cdot \mid \mathtt{A})$ *over* $G_\Sigma(\mathtt{A})$*.*

To continue our example, let $\mu(\mathtt{emacs(hmm1, tex)} \mid \mathtt{emacs(File', tex)}) = 0.4$ and $\mu(\mathtt{emacs(lohmm1, tex)} \mid \mathtt{emacs(File', tex)}) = 0.6$. Then there would be a probability of $0.4 \times 0.8 = 0.32$ that the next state is $\mathtt{emacs(hmm1, tex)}$ and of 0.48 that it is $\mathtt{emacs(lohmm1, tex)}$.

Taking μ into account, the meaning of an abstract transition $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ can be summarized as follows. Let $\mathtt{B}\theta_\mathtt{B} \in G_\Sigma(\mathtt{B})$, $\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H} \in G_\Sigma(\mathtt{H}\theta_\mathtt{B})$ and $\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}\theta_\mathtt{O} \in G_\Sigma(\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H})$. Then the model makes a transition from state $\mathtt{B}\theta_\mathtt{B}$ to $\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H}$ and emits symbol $\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}\theta_\mathtt{O}$ with probability

$$p \cdot \mu(\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H} \mid \mathtt{H}\theta_\mathtt{B}) \cdot \mu(\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}\theta_\mathtt{O} \mid \mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}). \tag{1}$$

To represent μ, any probabilistic representation can - in principle - be used, e.g. a Bayesian network or a Markov chain. Throughout the remainder of the present paper, however, we will use a *naïve Bayes* approach. More precisely, we associate to each argument of a relation $\mathtt{r}/m$ a finite domain $\mathrm{D}_i^{\mathtt{r}/m}$ of constants and a probability distribution $P_i^{\mathtt{r}/m}$ over $\mathrm{D}_i^{\mathtt{r}/m}$. Let $\mathrm{vars}(\mathtt{A}) = \{\mathtt{V}_1, \ldots, \mathtt{V}_l\}$ be the variables occurring in an atom $\mathtt{A}$ over $\mathtt{r}/m$, and let $\sigma = \{\mathtt{V}_1/\mathtt{s}_1, \ldots \mathtt{V}_l/\mathtt{s}_l\}$ be a substitution grounding $\mathtt{A}$. Each $\mathtt{V}_j$ is then considered a random variable over the domain $D_{\mathrm{arg}(\mathtt{V}_j)}^{\mathtt{r}/m}$ of the argument $\mathrm{arg}(\mathtt{V}_j)$ it appears first in. Then, $\mu(\mathtt{A}\sigma \mid \mathtt{A}) = \prod_{j=1}^{l} P_{\mathrm{arg}(\mathtt{V}_j)}^{\mathtt{r}/m}(\mathtt{s}_j)$. E.g. $\mu(\mathtt{emacs}(\mathtt{hmm1}, \mathtt{tex}) \mid \mathtt{emacs}(\mathtt{F}, \mathtt{E}))$, is computed as the product of $P_1^{\mathtt{emacs}/2}(\mathtt{hmm1})$ and $P_2^{\mathtt{emacs}/2}(\mathtt{tex})$.

Thus far the semantics of a single abstract transition has been defined. A LOHMM usually consists of multiple abstract transitions and this creates a further complication.

Example 3 *Consider* $0.8 : \mathtt{latex}(\mathtt{File}, \mathtt{tex}) \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs}(\mathtt{File}, \mathtt{tex})$ *and* $0.4 : \mathtt{dvi}(\mathtt{File}) \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs}(\mathtt{File}, \mathtt{User})$. *These two abstract transitions make conflicting statements about the state resulting from* $\mathtt{emacs}(\mathtt{hmm1}, \mathtt{tex})$. *Indeed, according to the first transition, the probability is* 0.8 *that the resulting state is* $\mathtt{latex}(\mathtt{hmm1}, \mathtt{tex})$ *and according to the second one it assigns* 0.4 *to* $\mathtt{xdvi}(\mathtt{hmm1})$.

There are essentially two ways to deal with this situation. On the one hand, one might want to combine and normalize the two transitions and assign a probability of $\frac{2}{3}$ respectively $\frac{1}{3}$. On the other hand, one might want to have only one rule firing. In this paper, we chose the latter option because it allows us to consider transitions more independently, it simplifies learning, and it yields locally interpretable models. We employ the subsumption (or generality) relation among the $\mathtt{B}$-parts of the two abstract transitions. Indeed, the $\mathtt{B}$-part of the first transition $\mathtt{B}_1 = \mathtt{emacs}(\mathtt{File}, \mathtt{tex})$ is more specific than that of the second transition $\mathtt{B}_2 = \mathtt{emacs}(\mathtt{File}, \mathtt{User})$ because there exists a substitution $\theta = \{\mathtt{User}/\mathtt{tex}\}$ such that $\mathtt{B}_2\theta = \mathtt{B}_1$, i.e., $\mathtt{B}_2$ subsumes $\mathtt{B}_1$. Therefore $G_{\Sigma_1}(\mathtt{B}_1) \subseteq G_{\Sigma_1}(\mathtt{B}_2)$ and the first transition can be regarded as more informative than the second one. It should therefore be preferred over the second one when starting from $\mathtt{emacs}(\mathtt{hmm1}, \mathtt{tex})$. We will also say that the first *transition* is *more specific* than the second one. Remark that this *generality* relation imposes a partial order on the set of all transitions. These considerations lead to the strategy of only considering the maximally specific transitions that apply to a state in order to determine the successor states. This implements a kind of exception handling or default reasoning and is akin to Katz's (1987) *back-off* n-gram models. In back-off n-gram models, the most detailed model that is deemed to provide sufficiently reliable information about the current context is used. That is, if one encounters an n-gram that is not sufficiently reliable, then back-off to use an $(n-1)$-gram; if that is not reliable either then back-off to level $n-2$, etc.

The conflict resolution strategy will work properly provided that the bodies of all maximally specific transitions (matching a given state) represent the same abstract state. This

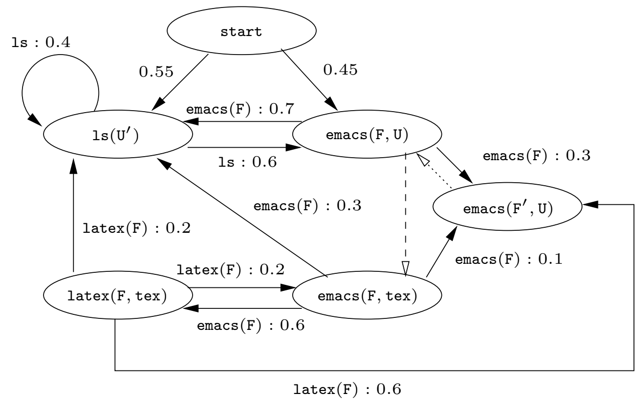


Figure 1: A logical hidden Markov model.

can be enforced by requiring the *generality* relation over the $\mathtt{B}$-parts to be closed under the *greatest lower bound* (glb) for each predicate, i.e., for each pair $\mathtt{B}_1, \mathtt{B}_2$ of bodies, such that $\theta = \mathrm{mgu}(\mathtt{B}_1, \mathtt{B}_2)$ exists, there is another body $\mathtt{B}$ (called lower bound) which subsumes $\mathtt{B}_1\theta$ (therefore also $\mathtt{B}_2\theta$) and is subsumed by $\mathtt{B}_1, \mathtt{B}_2$, and if there is any other lower bound then it is subsumed by $\mathtt{B}$. E.g., if the body of the second abstract transition in our example is $\mathtt{emacs(hmm1, User)}$ then the set of abstract transitions would not be closed under glb.

Finally, in order to specify a prior distribution over states, we assume a finite set Υ of clauses of the form $p : \mathtt{H} \leftarrow \mathtt{start}$ using a distinguished $\mathtt{start}$ symbol such that p is the probability of the LOHMM to start in a state of $G_\Sigma(\mathtt{H})$.

By now we are able to formally define *logical hidden Markov models.*

Definition 3 *A logical hidden Markov model (LOHMM) is a tuple* $(\Sigma, \mu, \Delta, \Upsilon)$ *where* Σ *is a logical alphabet,* μ *a selection probability over* Σ*,* Δ *is a set of abstract transitions, and* Υ *is a set of abstract transitions encoding a prior distribution. Let* $\mathbf{B}$ *be the set of all atoms that occur as body parts of transitions in* Δ*. We assume* $\mathbf{B}$ *to be closed under glb and require*

$$\forall \mathtt{B} \in \mathbf{B} : \sum\nolimits_{p:\mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta} p = 1.0 \tag{2}$$

and that the probabilities p *of clauses in* Υ *sum up to* 1.0 *.*

HMMs are a special cases of LOHMMs in which Σ contains only relation symbols of arity zero and the selection probability is irrelevant. Thus, LOHMMs directly generalize HMMs.

LOHMMs can also be represented graphically. Figure 1 contains an example. The underlying language Σ_2 consists of Σ_1 together with the constant symbol $\mathtt{other}$ which denotes a user that does not employ LaTeX. In this graphical notation, nodes represent abstract states and *black tipped arrows* denote abstract transitions. *White tipped arrows* are used to represent meta knowledge. More precisely, *white tipped, dashed arrows* represent the generality or subsumption ordering between abstract states. If we follow a transition to an abstract state with an outgoing *white tipped, dotted arrow* then this dotted arrow will always be followed. Dotted arrows are needed because the same abstract state can occur under different circumstances. Consider the transition $p : \mathtt{latex(File', User')} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, User)}$.

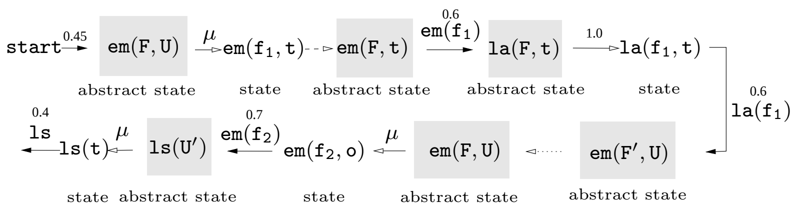


Figure 2: Generating the observation sequence `emacs(hmm1), latex(hmm1), emacs(lohmm1), ls` by the LOHMM in Figure 1. The command `emacs` is abbreviated by `em`, $\mathtt{f}_1$ denotes the filename `hmm1`, $\mathtt{f}_2$ represents `lohmm1`, `t` denotes a `tex` user, and `o` some `other` user. White tipped solid arrows indicate selections.

Even though the atoms in the head and body of the transition are syntactically different they represent the same abstract state. To accurately represent the meaning of this transition we cannot use a black tipped arrow from `latex(File, User)` to itself, because this would actually represent the abstract transition $p : \mathtt{latex(File, User)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, User)}$.

Furthermore, the graphical representation clarifies that LOHMMs are generative models. Let us explain how the model in Figure 1 would generate the observation sequence `emacs(hmm1), latex(hmm1), emacs(lohmm1), ls` (cf. Figure 2). It chooses an initial abstract state, say `emacs(F, U)`. Since both variables `F` and `U` are uninstantiated, the model samples the state `emacs(hmm1, tex)` from G_{Σ_2} using μ. As indicated by the dashed arrow, `emacs(F, tex)` is more specific than `emacs(F, U)`. Moreover, `emacs(hmm1, tex)` matches `emacs(F, tex)`. Thus, the model enters `emacs(F, tex)`. Since the value of `F` was already instantiated in the previous abstract state, `emacs(hmm1, tex)` is sampled with probability 1.0. Now, the model goes over to `latex(F, tex)`, emitting `emacs(hmm1)` because the abstract observation `emacs(F)` is already fully instantiated. Again, since F was already instantiated, `latex(hmm1, tex)` is sampled with probability 1.0. Next, we move on to `emacs(F′, U)`, emitting `latex(hmm1)`. Variables `F′` and `U` in `emacs(F′, U)` were not yet bound; so, values, say `lohmm1` and `others`, are sampled from μ. The dotted arrow brings us back to `emacs(F, U)`. Because variables are implicitly universally quantified in abstract transitions, the scope of variables is restricted to single abstract transitions. In turn, F is treated as a distinct, new variable, and is automatically unified with `F′`, which is bound to `lohmm1`. In contrast, variable `U` is already instantiated. Emitting `emacs(lohmm1)`, the model makes a transition to `ls(U′)`. Assume that it samples `tex` for `U′`. Then, it remains in `ls(U′)` with probability 0.4 . Considering all possible samples, allows one to prove the following theorem.

Theorem 1 (Semantics) *A logical hidden Markov model over a language* Σ *defines a discrete time stochastic process, i.e., a sequence of random variables* $\langle X_t \rangle_{t=1,2,\ldots}$*, where the domain of* X_t *is* $\mathrm{hb}(\Sigma) \times \mathrm{hb}(\Sigma)$*. The induced probability measure over the Cartesian product* $\bigotimes_t \mathrm{hb}(\Sigma) \times \mathrm{hb}(\Sigma)$ *exists and is unique for each* $t > 0$ *and in the limit* $t \to \infty$.

Before concluding this section, let us address some design choices underlying LOHMMs.

First, LOHMMs have been introduced as Mealy machines, i.e., output symbols are associated with transitions. Mealy machines fit our logical setting quite intuitively as they directly encode the conditional probability $P(\mathtt{O}, \mathtt{S}'|\mathtt{S})$ of making a transition from `S` to `S′`

emitting an observation $\mathtt{O}$. Logical hidden Markov models define this distribution as

$$P(\mathtt{O},\mathtt{S}'|\mathtt{S}) = \sum\nolimits_{p:\mathtt{H}\xleftarrow{\mathtt{O}'}\mathtt{B}} p \;\cdot\; \mu(\mathtt{S}' \mid \mathtt{H}\sigma_{\mathtt{B}}) \;\cdot\; \mu(\mathtt{O} \mid \mathtt{O}'\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})$$

where the sum runs over all abstract transitions $\mathtt{H} \xleftarrow{\mathtt{O}'} \mathtt{B}$ such that $\mathtt{B}$ is most specific for $\mathtt{S}$. Observations correspond to (partially) observed proof steps and, hence, provide information shared among heads and bodies of abstract transitions. In contrast, HMMs are usually introduced as *Moore* machines. Here, output symbols are associated with states implicitly assuming $\mathtt{O}$ and $\mathtt{S}'$ to be independent. Thus, $P(\mathtt{O},\mathtt{S}' \mid \mathtt{S})$ factorizes into $P(\mathtt{O} \mid \mathtt{S}) \cdot P(\mathtt{S}' \mid \mathtt{S})$. This makes it more difficult to observe information shared among heads and bodies. In turn, Moore-LOHMMs are less intuitive and harder to understand. For a more detailed discussion of the issue, we refer to Appendix B where we essentially show that – as in the propositional case – Mealy- and Moore-LOHMMs are equivalent.

Second, the naïve Bayes approach for the selection distribution reduces the model complexity at the expense of a lower expressivity: functors are neglected and variables are treated independently. Adapting more expressive approaches is an interesting future line of research. For instance, *Bayesian networks* allow one to represent *factorial HMMs* (Ghahramani & Jordan, 1997). Factorial HMMs can be viewed as LOHMMs, where the hidden states are summarized by a $2 \cdot k$-ary abstract state. The first k arguments encode the k state variables, and the last k arguments serve as a memory of the previous joint state. μ of the i-th argument is conditioned on the $i + k$-th argument. *Markov chains* allow one to sample compound terms of finite depth such as $\mathtt{s(s(s(0)))}$ and to model e.g. misspelled filenames. This is akin to *generalized HMMs* (Kulp, Haussler, Reese, & Eeckman, 1996), in which each node may output a finite sequence of symbols rather than a single symbol.

Finally, LOHMMs – as introduced in the present paper – specify a probability distribution over all sequences of a given length. Reconsider the LOHMM in Figure 1. Already the probabilities of all observation sequences of length 1, i.e., $\mathtt{ls}$, $\mathtt{emacs(hmm1)}$, and $\mathtt{emacs(lohmm1)}$) sum up to 1. More precisely, for each $t > 0$ it holds that $\sum_{x_1,\ldots,x_t} P(X_1 = x_1,\ldots,X_t = x_t) = 1.0$. In order to model a distribution over sequences of variable length, i.e., $\sum_{t>0}\sum_{x_1,\ldots,x_t} P(X_1 = x_1,\ldots,X_t = x_t) = 1.0$ we may add a distinguished $\mathtt{end}$ state. The $\mathtt{end}$ state is absorbing in that whenever the model makes a transition into this state, it terminates the observation sequence generated.

4. Three Inference Problems for LOHMMs

As for HMMs, three inference problems are of interest. Let M be a LOHMM and let $\mathsf{O} = \mathtt{O}_1, \mathtt{O}_2, \ldots, \mathtt{O}_T$, $T > 0$, be a finite sequence of ground observations:

(1) Evaluation: Determine the probability $P(\mathsf{O} \mid M)$ that sequence O was generated by the model M.

(2) Most likely state sequence: Determine the hidden state sequence S^* that has most likely produced the observation sequence O, i.e. $\mathsf{S}^* = \arg\max_{\mathsf{S}} P(\mathsf{S} \mid \mathsf{O}, M)$.

(3) Parameter estimation: Given a set $\boldsymbol{\mathsf{O}} = \{\mathsf{O}_1, \ldots, \mathsf{O}_k\}$ of observation sequences, determine the most likely parameters λ^* for the abstract transitions and the selection distribution of M, i.e. $\lambda^* = \arg\max_{\lambda} P(\boldsymbol{\mathsf{O}} \mid \lambda)$.

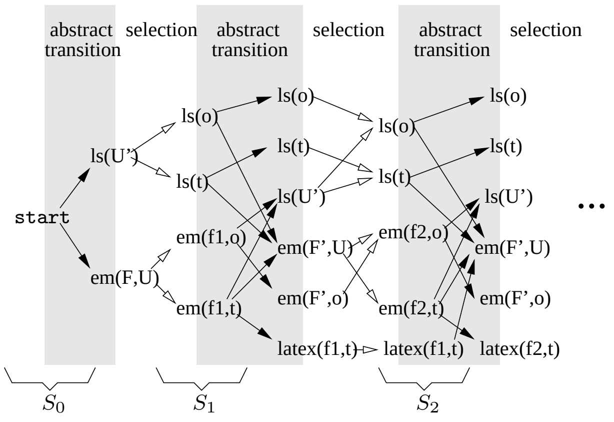


Figure 3: Trellis induced by the LOHMM in Figure 1. The sets of reachable states at time $0, 1, \ldots$ are denoted by $S_0, S_1, \ldots$ In contrast with HMMs, there is an additional layer where the states are sampled from abstract states.

We will now address each of these problems in turn by upgrading the existing solutions for HMMs. This will be realized by computing a grounded trellis as in Figure 3. The possible ground successor states of any given state are computed by first selecting the applicable abstract transitions and then applying the selection probabilities (while taking into account the substitutions) to ground the resulting states. This two-step factorization is coalesced into one step for HMMs.

To **evaluate** $\mathtt{O}$, consider the probability of the partial observation sequence $\mathtt{O}_1, \mathtt{O}_2, \ldots, \mathtt{O}_t$ and (ground) state $\mathtt{S}$ at time t, $0 < t \leq T$, given the model $M = (\Sigma, \mu, \Delta, \Upsilon)$

$$\alpha_t(\mathtt{S}) := P(\mathtt{O}_1, \mathtt{O}_2, \ldots, \mathtt{O}_t, q_t = \mathtt{S} \mid M)$$

where $q_t = \mathtt{S}$ denotes that the system is in state $\mathtt{S}$ at time t. As for HMMs, $\alpha_t(\mathtt{S})$ can be computed using a dynamic programming approach. For $t = 0$, we set $\alpha_0(\mathtt{S}) = P(q_0 = \mathtt{S} \mid M)$, i.e., $\alpha_0(\mathtt{S})$ is the probability of starting in state $\mathtt{S}$ and, for $t > 0$, we compute $\alpha_t(\mathtt{S})$ based on $\alpha_{t-1}(\mathtt{S}')$:

```
1:  $S_0 := \{\mathtt{start}\}$        /* initialize the set of reachable states*/
2:  for $t = 1, 2, \ldots, T$ do
3:      $S_t = \emptyset$        /* initialize the set of reachable states at clock t*/
4:      foreach $\mathtt{S} \in S_{t-1}$ do
5:          $\boxed{\text{foreach maximally specific } p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta \cup \Upsilon \text{ s.t. } \sigma_\mathtt{B} = \mathrm{mgu}(\mathtt{S}, \mathtt{B}) \text{ exists do}}$
6:              $\boxed{\text{foreach } \mathtt{S}' = \mathtt{H}\sigma_\mathtt{B}\sigma_\mathtt{H} \in G_\Sigma(\mathtt{H}\sigma_\mathtt{B}) \text{ s.t. } \mathtt{O}_{t-1} \text{ unifies with } \mathtt{O}\sigma_\mathtt{B}\sigma_\mathtt{H} \text{ do}}$
7:                  if $\mathtt{S}' \notin S_t$ then
8:                      $S_t := S_t \cup \{\mathtt{S}'\}$
9:                      $\alpha_t(\mathtt{S}') := 0.0$
10:                 $\alpha_t(\mathtt{S}') := \alpha_t(\mathtt{S}') + \alpha_{t-1}(\mathtt{S}) \cdot p \cdot \boxed{\mu(\mathtt{S}' \mid \mathtt{H}\sigma_\mathtt{B}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_\mathtt{B}\sigma_\mathtt{H})}$
11: return $P(\mathtt{O} \mid M) = \sum_{\mathtt{S} \in S_T} \alpha_T(\mathtt{S})$
```

where we assume for the sake of simplicity $\texttt{O} \equiv \texttt{start}$ for each abstract transition $p : \texttt{H} \leftarrow \texttt{start} \in \Upsilon$. Furthermore, the boxed parts specify all the differences to the HMM formula: unification and μ are taken into account.

Clearly, as for HMMs $P(\texttt{O} \mid M) = \sum_{\texttt{S} \in S_T} \alpha_T(\texttt{S})$ holds. The computational complexity of this *forward procedure* is $\mathcal{O}(T \cdot s \cdot (|\mathbf{B}| + o \cdot g)) = \mathcal{O}(T \cdot s^2)$ where $s = \max_{t=1,2,\ldots,T} |S_t|$, o is the maximal number of outgoing abstract transitions with regard to an abstract state, and g is the maximal number of ground instances of an abstract state. In a completely analogous manner, one can devise a *backward procedure* to compute

$$\beta_t(\texttt{S}) = P(\texttt{O}_{t+1}, \texttt{O}_{t+2}, \ldots, \texttt{O}_T \mid q_t = \texttt{S}, M) .$$

This will be useful for solving Problem **(3)**.

Having a forward procedure, it is straightforward to adapt the Viterbi algorithm as a solution to Problem **(2)**, i.e., for computing the **most likely state sequence**. Let $\delta_t(\texttt{S})$ denote the highest probability along a single path at time t which accounts for the first t observations and ends in state $\texttt{S}$, i.e.,

$$\delta_t(\texttt{S}) = \max_{\texttt{S}_0, \texttt{S}_1, \ldots, \texttt{S}_{t-1}} P(\texttt{S}_0, \texttt{S}_1, \ldots, \texttt{S}_{t-1}, \texttt{S}_t = \texttt{S}, O_1, \ldots, O_{t-1} | M) .$$

The procedure for finding the most likely state sequence basically follows the forward procedure. Instead of summing over all ground transition probabilities in line 10, we maximize over them. More precisely, we proceed as follows:

```
1: $S_0 := \{\texttt{start}\}$                /* initialize the set of reachable states*/
2: for $t = 1, 2, \ldots, T$ do
3:     $S_t = \emptyset$                 /* initialize the set of reachable states at clock t*/
4:     foreach $\texttt{S} \in S_{t-1}$ do
5:         foreach maximally specific $p : \texttt{H} \xleftarrow{\texttt{O}} \texttt{B} \in \Delta \cup \Upsilon$ s.t. $\sigma_\texttt{B} = \text{mgu}(\texttt{S}, \texttt{B})$ exists do
6:             foreach $\texttt{S}' = \texttt{H}\sigma_\texttt{B}\sigma_\texttt{H} \in G_\Sigma(\texttt{H}\sigma_\texttt{B})$ s.t. $\texttt{O}_{t-1}$ unifies with $\texttt{O}\sigma_\texttt{B}\sigma_\texttt{H}$ do
7:                 if $\texttt{S}' \notin S_t$ then
8:                     $S_t := S_t \cup \{\texttt{S}'\}$
9:                     $\delta_t(\texttt{S}, \texttt{S}') := 0.0$
10:                $\delta_t(\texttt{S}, \texttt{S}') := \delta_t(\texttt{S}, \texttt{S}') + \delta_{t-1}(\texttt{S}) \cdot p \cdot \mu(\texttt{S}' \mid \texttt{H}\sigma_\texttt{B}) \cdot \mu(\texttt{O}_{t-1} \mid \texttt{O}\sigma_\texttt{B}\sigma_\texttt{H})$
11:    foreach $\texttt{S}' \in S_t$ do
12:        $\delta_t(\texttt{S}') = \max_{\texttt{S} \in S_{t-1}} \delta_t(\texttt{S}, \texttt{S}')$
13:        $\psi_t(\texttt{S}') = \arg\max_{\texttt{S} \in S_{t-1}} \psi_t(\texttt{S}, \texttt{S}')$
```

Here, $\delta_t(\texttt{S}, \texttt{S}')$ stores the probability of making a transition from $\texttt{S}$ to $\texttt{S}'$ and $\psi_t(\texttt{S}')$ (with $\psi_1(S) = \texttt{start}$ for all states S) keeps track of the state maximizing the probability along a single path at time t which accounts for the first t observations and ends in state $\texttt{S}'$. The most likely hidden state sequence $\texttt{S}^*$ can now be computed as

$$\begin{aligned} \texttt{S}^*_{T+1} &= \arg\max_{\texttt{S} \in S_{T+1}} \delta_{T+1}(\texttt{S}) \\ \text{and } \texttt{S}^*_t &= \psi_t(\texttt{S}^*_{t+1}) \text{ for } t = T, T-1, \ldots, 1 . \end{aligned}$$

One can also consider problem **(2)** on a more abstract level. Instead of considering all contributions of different abstract transitions $\texttt{T}$ to a single ground transition from state $\texttt{S}$

to state $\mathtt{S}'$ in line 10, one might also consider the most likely abstract transition only. This is realized by replacing line 10 in the forward procedure with

$$\alpha_t(\mathtt{S}') := \max(\alpha_t(\mathtt{S}'), \alpha_{t-1}(\mathtt{S}) \cdot p \cdot \ \mu(\mathtt{S}' \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}}))\ .$$

This solves the problem of finding the **(2′) most likely state and abstract transition sequence**:

> Determine the sequence of states and abstract transitions $\mathsf{GT}^* = S_0, \mathtt{T}_0, S_1, \mathtt{T}_1, \mathtt{S}_2, \ldots, \mathtt{S}_\mathtt{T}, \mathtt{T}_T, \mathtt{S}_{\mathtt{T}+1}$ where there exists substitutions θ_i with $S_{i+1} \leftarrow S_i \equiv \mathtt{T}_i\theta_i$ that has most likely produced the observation sequence O, i.e. $\mathsf{GT}^* = \arg\max_{\mathsf{GT}} P(\mathsf{GT} \mid \mathsf{O}, M)$.

Thus, logical hidden Markov models also pose new types of inference problems.

For **parameter estimation**, we have to estimate the maximum likelihood transition probabilities and selection distributions. To estimate the former, we upgrade the well-known *Baum-Welch* algorithm (Baum, 1972) for estimating the maximum likelihood parameters of HMMs and probabilistic context-free grammars.

For HMMs, the Baum-Welch algorithm computes the improved estimate $\overline{p}$ of the transition probability of some (ground) transition $\mathtt{T} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ by taking the ratio

$$\overline{p} = \frac{\xi(\mathtt{T})}{\sum_{\mathtt{H}' \xleftarrow{\mathtt{O}'} \mathtt{B} \in \Delta \cup \Upsilon} \xi(\mathtt{T}')} \tag{3}$$

between the expected number $\xi(\mathtt{T})$ of times of making the transitions $\mathtt{T}$ at any time given the model M and an observation sequence O, and the total number of times a transitions is made from $\mathtt{B}$ at any time given M and O.

Basically the same applies when $\mathtt{T}$ is an abstract transition. However, we have to be a little bit more careful because we have no direct access to $\xi(\mathtt{T})$. Let $\xi_t(\text{gcl}, \mathtt{T})$ be the probability of following the abstract transition $\mathtt{T}$ via its ground instance $\text{gcl} \equiv p : \mathtt{GH} \xleftarrow{\mathtt{GO}} \mathtt{GB}$ at time t, i.e.,

$$\xi_t(\text{gcl}, \mathtt{T}) = \frac{\alpha_t(\mathtt{GB}) \cdot p \cdot \beta_{t+1}(\mathtt{GH})}{P(\mathsf{O} \mid M)} \cdot \boxed{\mu(\mathtt{GH} \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})}\ , \tag{4}$$

where $\sigma_{\mathtt{B}}, \sigma_{\mathtt{H}}$ are as in the forward procedure (see above) and $P(\mathsf{O} \mid M)$ is the probability that the model generated the sequence O. Again, the boxed terms constitute the main difference to the corresponding HMM formula. In order to apply Equation (3) to compute improved estimates of probabilities associated with abstract transitions, we set

$$\xi(\mathtt{T}) = \sum_{t=1}^{T} \xi_t(\mathtt{T}) = \sum_{t=1}^{T} \sum_{\text{gcl}} \xi_t(\text{gcl}, \mathtt{T})$$

where the inner sum runs over all ground instances of $\mathtt{T}$.

This leads to the following re-estimation method, where we assume that the sets S_i of reachable states are reused from the computations of the α- and β-values:

```
1: /* initialization of expected counts */
2: foreach $\mathtt{T} \in \Delta \cup \Upsilon$ do
3:     $\xi(\mathtt{T}) := m$   /* or 0 if not using pseudocounts */
4: /* compute expected counts */
5: for $t = 0, 1, \ldots, T$ do
6:     foreach $\mathtt{S} \in S_t$ do
7:         $\boxed{\textbf{foreach} \text{ max. specific } \mathtt{T} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta \cup \Upsilon \text{ s.t. } \sigma_{\mathtt{B}} = \mathrm{mgu}(\mathtt{S}, \mathtt{B}) \text{ exists } \textbf{do}}$
8:             $\boxed{\textbf{foreach} \ \mathtt{S}' = \mathtt{H}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}} \in G_{\Sigma}(\mathtt{H}\sigma_{\mathtt{B}}) \text{ s.t. } \mathtt{S}' \in S_{t+1} \wedge \mathrm{mgu}(\mathtt{O}_t, \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}}) \text{ exists } \textbf{do}}$
9:                 $\xi(\mathtt{T}) := \xi(\mathtt{T}) + \alpha_t(\mathtt{S}) \cdot p \cdot \beta_{t+1}(\mathtt{S}') / P(\mathtt{O} \mid M) \cdot \boxed{\mu(\mathtt{S}' \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})}$
```

Here, equation (4) can be found in line 9. In line 3, we set pseudocounts as small sample-size regularizers. Other methods to avoid a biased underestimate of probabilities and even zero probabilities such as m-estimates (see e.g., Mitchell, 1997) can be easily adapted.

To estimate the selection probabilities, recall that μ follows a naïve Bayes scheme. Therefore, the estimated probability for a domain element $d \in D$ for some domain D is the ratio between the number of times d is selected and the number of times any $d' \in D$ is selected. The procedure for computing the ξ-values can thus be reused.

Altogether, the Baum-Welch algorithm works as follows: While not converged, (1) estimate the abstract transition probabilities, and (2) the selection probabilities. Since it is an instance of the EM algorithm, it increases the likelihood of the data with every update, and according to McLachlan and Krishnan (1997), it is guaranteed to reach a stationary point. All standard techniques to overcome limitations of EM algorithms are applicable. The computational complexity (per iteration) is $\mathcal{O}(k \cdot (\alpha + d)) = \mathcal{O}(k \cdot T \cdot s^2 + k \cdot d)$ where k is the number of sequences, α is the complexity of computing the α-values (see above), and d is the sum over the sizes of domains associated to predicates. Recently, Kersting and Raiko (2005) combined the Baum-Welch algorithm with structure search for model selection of logical hidden Markov models using *inductive logic programming* (Muggleton & De Raedt, 1994) refinement operators. The refinement operators account for different abstraction levels which have to be explored.

5. Advantages of LOHMMs

In this section, we will investigate the benefits of LOHMMs: **(1)** LOHMMs are strictly more expressive than HMMs, and **(2)**, using abstraction, logical variables and unification can be beneficial. More specifically, with **(2)**, we will show that

(B1) LOHMMs can be — by design — smaller than their propositional instantiations, and

(B2) unification can yield better log-likelihood estimates.

5.1 On the Expressivity of LOHMMs

Whereas HMMs specify probability distributions over regular languages, LOHMMs specify probability distributions over more expressive languages.

Theorem 2 *For any (consistent) probabilistic context-free grammar (PCFG) G for some language $\mathcal{L}$ there exists a LOHMM M s.t. $P_G(w) = P_M(w)$ for all $w \in \mathcal{L}$.*

The proof (see Appendix C) makes use of abstract states of unbounded 'depth'. More precisely, functors are used to implement a stack. Without functors, LOHMMs cannot encode PCFGs and, because the Herbrand base is finite, it can be proven that there always exists an equivalent HMM.

Furthermore, if functors are allowed, LOHMMs are strictly more expressive than PCFGs. They can specify probability distributions over some languages that are context-sensitive:

$$\begin{array}{lrcl}
1.0: & \mathtt{stack(s(0),s(0))} & \leftarrow & \mathtt{start} \\
0.8: & \mathtt{stack(s(X),s(X))} & \xleftarrow{\mathtt{a}} & \mathtt{stack(X,X)} \\
0.2: & \mathtt{unstack(s(X),s(X))} & \xleftarrow{\mathtt{a}} & \mathtt{stack(X,X)} \\
1.0: & \mathtt{unstack(X,Y)} & \xleftarrow{\mathtt{b}} & \mathtt{unstack(s(X),Y)} \\
1.0: & \mathtt{unstack(s(0),Y)} & \xleftarrow{\mathtt{c}} & \mathtt{unstack(s(0),s(Y))} \\
1.0: & \mathtt{end} & \xleftarrow{\mathtt{end}} & \mathtt{unstack(s(0),s(0))}
\end{array}$$

The LOHMM defines a distribution over $\{a^n b^n c^n \mid n > 0\}$.

Finally, the use of logical variables also enables one to deal with *identifiers*. Identifiers are special types of constants that denote objects. Indeed, recall the UNIX command sequence `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, ... from the introduction. The filename `lohmms.tex` is an identifier. Usually, the specific identifiers do not matter but rather the fact that the same object occurs multiple times in the sequence. LOHMMs can easily deal with identifiers by setting the selection probability μ to a constant for the arguments in which identifiers can occur. Unification then takes care of the necessary variable bindings.

5.2 Benefits of Abstraction through Variables and Unification

Reconsider the domain of UNIX command sequences. Unix users oftenly reuse a newly created directory in subsequent commands such as in `mkdir(vt100x)`, `cd(vt100x)`, `ls(vt100x)` . Unification should allow us to elegantly employ this information because it allows us to specify that, after observing the created directory, the model makes a transition into a state where the newly created directory is used:

$$p_1 : \mathtt{cd(Dir,mkdir)} \leftarrow \mathtt{mkdir(Dir,com)} \quad \text{and} \quad p_2 : \mathtt{cd(\_,mkdir)} \leftarrow \mathtt{mkdir(Dir,com)}$$

If the first transition is followed, the `cd` command will move to the newly created directory; if the second transition is followed, it is not specified which directory `cd` will move to. Thus, the LOHMM captures the reuse of created directories as an argument of future commands. Moreover, the LOHMM encodes the simplest possible case to show the benefits of unification. At any time, the observation sequence uniquely determines the state sequence, and functors are not used. Therefore, we left out the abstract output symbols associated with abstract transitions. In total, the LOHMM U, modelling the reuse of directories, consists of 542 parameters only but still covers more than 451000 (ground) states, see Appendix D for the complete model. The compression in the number of parameters supports **(B1)**.

To empirically investigate the benefits of unification, we compare U with the variant N of U where no variables are shared, i.e., no unification is used such that for instance the first transition above is not allowed, see Appendix D. N has 164 parameters less than U. We computed the following zero-one win function

$$f(\mathsf{O}) = \begin{cases} 1 & \text{if } \left[\log P_U(\mathsf{O}) - \log P_N(\mathsf{O})\right] > 0 \\ 0 & \text{otherwise} \end{cases}$$

leave-one-out cross-validated on Unix shell logs collected by Greenberg (1988). Overall, the data consists of 168 users of four groups: computer scientists, nonprogrammers, novices and others. About 300000 commands have been logged with an average of 110 sessions per user. We present here results for a subset of the data. We considered all computer scientist sessions in which at least a single `mkdir` command appears. These yield 283 logical sequences over in total 3286 ground atoms. The LOO win was 81.63%. Other LOO statistics are also in favor of U:

	training		test	
	$\log P(\mathbf{O})$	$\log \frac{P_U(\mathbf{O})}{P_N(\mathbf{O})}$	$\log P(\mathsf{O})$	$\log \frac{P_U(\mathsf{O})}{P_N(\mathsf{O})}$
U	-11361.0	1795.3	-42.8	7.91
N	-13157.0		-50.7	

Thus, although U has 164 parameters more than N, it shows a better generalization performance. This result supports **(B2)**. A pattern often found in U was [1]

$$0.15 : \texttt{cd(Dir,mkdir)} \leftarrow \texttt{mkdir(Dir,com)} \quad \text{and} \quad 0.08 : \texttt{cd(\_,mkdir)} \leftarrow \texttt{mkdir(Dir,com)}$$

favoring changing to the directory just made. This knowledge cannot be captured in N

$$0.25 : \quad \texttt{cd(\_,mkdir)} \quad \leftarrow \quad \texttt{mkdir(Dir,com)}.$$

The results clearly show that abstraction through variables and unification can be beneficial for some applications, i.e., **(B1)** and **(B2)** hold.

6. Real World Applications

Our intentions here are to investigate whether LOHMMs can be applied to real world domains. More precisely, we will investigate whether benefits **(B1)** and **(B2)** can also be exploited in real world application domains. Additionally, we will investigate whether

(B3) LOHMMs are competitive with ILP algorithms that can also utilize unification and abstraction through variables, and

(B4) LOHMMs can handle tree-structured data similar to PCFGs.

To this aim, we conducted experiments on two bioinformatics application domains: protein fold recognition (Kersting, Raiko, Kramer, & De Raedt, 2003) and mRNA signal structure detection (Horváth, Wrobel, & Bohnebeck, 2001). Both application domains are multiclass problems with five different classes each.

1. The sum of probabilities is not the same ($0.15 + 0.08 = 0.23 \neq 0.25$) because of the use of pseudo counts and because of the subliminal non-determinism (w.r.t. abstract states) in U, i.e., in case that the first transition fires, the second one also fires.

6.1 Methodology

In order to tackle the multiclass problem with LOHMMs, we followed a *plug-in estimate* approach. Let $\{c_1, c_2, \ldots, c_k\}$ be the set of possible classes. Given a finite set of training examples $\{(x_i, y_i)\}_{i=1}^n \subseteq \mathcal{X} \times \{c_1, c_2, \ldots, c_n\}$, one tries to find $f : \mathcal{X} \to \{c_1, c_2, \ldots, c_k\}$

$$f(x) = \arg \max_{c \in \{c_1, c_2, \ldots, c_k\}} P(x \mid M, \lambda_c^*) \cdot P(c) \; . \tag{5}$$

with low approximation error on the training data as well as on unseen examples. In Equation (5), M denotes the model structure which is the same for all classes, λ_c^* denotes the maximum likelihood parameters of M for class c estimated on the training examples with $y_i = c$ only, and $P(c)$ is the prior class distribution.

We implemented the Baum-Welch algorithm (with pseudocounts m, see line 3) for maximum likelihood parameter estimation using the Prolog system Yap-4.4.4. In all experiments, we set $m = 1$ and let the Baum-Welch algorithm stop if the change in log-likelihood was less than 0.1 from one iteration to the next. The experiments were ran on a Pentium-IV 3.2 GHz Linux machine.

6.2 Protein Fold Recognition

Protein fold recognition is concerned with how proteins fold in nature, i.e., their three-dimensional structures. This is an important problem as the biological functions of proteins depend on the way they fold. A common approach is to use database searches to find proteins (of known fold) similar to a newly discovered protein (of unknown fold). To facilitate protein fold recognition, several expert-based classification schemes of proteins have been developed that group the current set of known protein structures according to the similarity of their folds. For instance, the *structural classification of proteins* (Hubbard, Murzin, Brenner, & Chotia, 1997) (SCOP) database hierarchically organizes proteins according to their structures and evolutionary origin. From a machine learning perspective, SCOP induces a classification problem: given a protein of unknown fold, assign it to the best matching group of the classification scheme. This *protein fold classification* problem has been investigated by Turcotte, Muggleton, and Sternberg (2001) based on the inductive logic programming (ILP) system PROGOL and by Kersting et al. (2003) based on LOHMMs.

The secondary structure of protein domains[2] can elegantly be represented as logical sequences. For example, the secondary structure of the Ribosomal protein L4 is represented as

$$\begin{aligned}&\texttt{st}(\texttt{null}, 2), \texttt{he}(\texttt{right}, \texttt{alpha}, 6), \texttt{st}(\texttt{plus}, 2), \texttt{he}(\texttt{right}, \texttt{alpha}, 4), \texttt{st}(\texttt{plus}, 2),\\ &\texttt{he}(\texttt{right}, \texttt{alpha}, 4), \texttt{st}(\texttt{plus}, 3), \texttt{he}(\texttt{right}, \texttt{alpha}, 4), \texttt{st}(\texttt{plus}, 1), \texttt{he}(\texttt{hright}, \texttt{alpha}, 6)\end{aligned}$$

Helices of a certain type, orientation and length $\texttt{he}(\mathit{HelixType}, \mathit{HelixOrientation}, \mathit{Length})$, and strands of a certain orientation and length $\texttt{st}(\mathit{StrandOrientation}, \mathit{Length})$ are atoms over logical predicates. The application of traditional HMMs to such sequences requires one to either ignore the structure of helices and strands, which results in a loss of information, or to take all possible combinations (of arguments such as orientation and length) into account, which leads to a combinatorial explosion in the number of parameters

2. A domain can be viewed as a sub-section of a protein which appears in a number of distantly related proteins and which can fold independently of the rest of the protein.

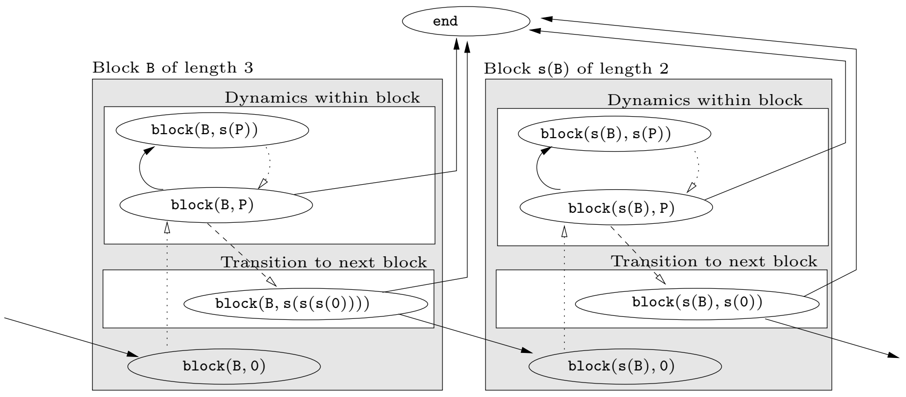
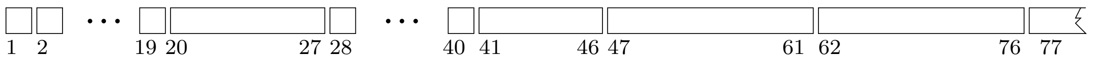


Figure 4: Scheme of a left-to-right LOHMM block model.

The results reported by Kersting et al. (2003) indicate that LOHMMs are well-suited for protein fold classification: the number of parameters of a LOHMM can by an order of magnitude be smaller than the number of a corresponding HMM (120 versus approximately 62000) and the generalization performance, a 74% accuracy, is comparable to Turcotte et al.'s (2001) result based on the ILP system Progol, a 75% accuracy. Kersting et al. (2003), however, do not cross-validate their results nor investigate – as it is common in bioinformatics – the impact of primary sequence similarity on the classification accuracy. For instance, the two most commonly requested ASTRAL subsets are the subset of sequences with less than 95% identity to each other (95 cut) and with less than 40% identity to each other (40 cut). Motivated by this, we conducted the following new experiments.

The data consists of logical sequences of the secondary structure of protein domains. As in the work of Kersting et al. (2003), the task is to predict one of the five most populated SCOP folds of alpha and beta proteins (a/b): TIM beta/alpha-barrel (fold 1), NAD(P)-binding Rossmann-fold domains (fold 2), Ribosomal protein L4 (fold 23), Cysteine hydrolase (fold 37), and Phosphotyrosine protein phosphatases I-like (fold 55). The class of a/b proteins consists of proteins with mainly parallel beta sheets (beta-alpha-beta units). The data have been extracted automatically from the ASTRAL dataset version 1.65 (Chandonia, Hon, Walker, Lo Conte, P.Koehl, & Brenner, 2004) for the 95 cut and for the 40 cut. As in the work of Kersting et al. (2003), we consider strands and helices only, i.e., coils and isolated strands are discarded. For the 95 cut, this yields 816 logical sequences consisting of in total 22210 ground atoms. The number of sequences in the classes are listed as 293, 151, 87, 195, and 90. For the 40 cut, this yields 523 logical sequences consisting of in total 14986 ground atoms. The number of sequences in the classes are listed as 182, 100, 66, 122, and 53.

LOHMM structure: The used LOHMM structure follows a left-to-right block topology, see Figure 4, to model blocks of consecutive helices (resp. strands). Being in a *Block* of some size s, say 3, the model will remain in the same block for $s = 3$ time steps. A similar idea has been used to model haplotypes (Koivisto, Perola, Varilo, Hennah, Ekelund, Lukk, Peltonen, Ukkonen, & Mannila, 2002; Koivisto, Kivioja, Mannila, Rastas, & Ukkonen, 2004). In contrast to common HMM block models (Won, Prügel-Bennett, & Krogh, 2004), the transition parameters are shared within each block and one can ensure that the model makes a transition to the next state s(*Block*) only at the end of a block; in our example after exactly 3 intra-block transitions. Furthermore, there are specific abstract transitions for all helix types and strand orientations to model the priori distribution, the intra- and the inter-block transitions. The number of blocks and their sizes were chosen according to the empirical distribution over sequence lengths in the data so that the beginning and the ending of protein domains was likely captured in detail. This yield the following block structure



where the numbers denote the positions within protein domains. Furthermore, note that the last block gathers all remaining transitions. The blocks themselves are modelled using hidden abstract states over

$$\texttt{hc}(\mathit{HelixType}, \mathit{HelixOrientation}, \mathit{Length}, \mathit{Block}) \text{ and } \texttt{sc}(\mathit{StrandOrientation}, \mathit{Length}, \mathit{Block})\ .$$

Here, *Length* denotes the number of consecutive bases the structure element consists of. The length was discretized into 10 bins such that the original lengths were uniformally distributed. In total, the LOHMM has 295 parameters. The corresponding HMM without parameter sharing has more than 65200 parameters. This clearly confirms **(B1)**.

Results: We performed a 10-fold cross-validation. On the 95 cut dataset, the accuracy was 76% and took approx. 25 minutes per cross-validation iteration; on the 40 cut, the accuracy was 73% and took approx. 12 minutes per cross-validation iteration. The results validate Kersting et al.'s (2003) results and, in turn, clearly show that **(B3)** holds. Moreover, the novel results on the 40 cut dataset indicate that the similarities detected by the LOHMMs between the protein domain structures were not accompanied by high sequence similarity.

6.3 mRNA Signal Structure Detection

mRNA sequences consist of bases (guanine, adenine, uracil, cytosine) and fold intramolecularly to form a number of short base-paired stems (Durbin, Eddy, Krogh, & Mitchison, 1998). This base-paired structure is called the *secondary structure*, cf. Figures 5 and 6. The secondary structure contains special subsequences called signal structures that are responsible for special biological functions, such as RNA-protein interactions and cellular transport. The function of each signal structure class is based on the common characteristic binding site of all class elements. The elements are not necessarily identical but very similar. They can vary in topology (tree structure), in size (number of constituting bases), and in base sequence.

The goal of our experiments was to recognize instances of signal structures classes in mRNA molecules. The first application of relational learning to recognize the signal structure class of mRNA molecules was described in the works of Bohnebeck, Horváth, and Wrobel (1998) and of Horváth et al. (2001), where the relational instance-based learner RIBL was applied. The dataset [3] we used was similar to the one described by Horváth

3. The dataset is not the same as described in the work by Horváth et al. (2001) because we could not obtain the original dataset. We will compare to the smaller data set used by Horváth et al., which consisted of

et al. (2001). It consisted of 93 mRNA secondary structure sequences. More precisely, it was composed of 15 and 5 SECIS (Selenocysteine Insertion Sequence), 27 IRE (Iron Responsive Element), 36 TAR (Trans Activating Region) and 10 histone stem loops constituting five classes.

The secondary structure is composed of different building blocks such as stacking region, hairpin loops, interior loops etc. In contrast to the secondary structure of proteins that forms chains, the secondary structure of mRNA forms a tree. As trees can not easily be handled using HMMs, mRNA secondary structure data is more challenging than that of proteins. Moreover, Horváth et al. (2001) report that making the tree structure available to RIBL as background knowledge had an influence on the classification accuracy. More precisely, using a simple chain representation RIBL achieved a 77.2% leave-one-out cross-validation (LOO) accuracy whereas using the tree structure as background knowledge RIBL achieved a 95.4% LOO accuracy.

We followed Horváth et al.'s experimental setup, that is, we adapted their data representations to LOHMMs and compared a chain model with a tree model.

Chain Representation: In the *chain* representation (see also Figure 5), signal structures are described by $\texttt{single}(\mathit{TypeSingle}, \mathit{Position}, \mathit{Acid})$ or $\texttt{helical}(\mathit{TypeHelical}, \mathit{Position}, \mathit{Acid}, \mathit{Acid})$. Depending on its type, a structure element is represented by either $\texttt{single}/3$ or $\texttt{helical}/4$. Their first argument *TypeSingle* (resp. *TypeHelical*) specifies the type of the structure element, i.e., $\texttt{single}, \texttt{bulge3}, \texttt{bulge5}, \texttt{hairpin}$ (resp. $\texttt{stem}$). The argument *Position* is the position of the sequence element within the corresponding structure element counted down, i.e.[4], $\{\texttt{n}^{13}(\texttt{0}), \texttt{n}^{12}(\texttt{0}), \ldots, \texttt{n}^{1}(\texttt{0})\}$. The maximal position was set to 13 as this was the maximal position observed in the data. The last argument encodes the observed nucleotide (pair).

The used LOHMM structure follows again the left-to-right block structure shown in Figure 4. Its underlying idea is to model blocks of consecutive helical structure elements. The hidden states are modelled using $\texttt{single}(\mathit{TypeSingle}, \mathit{Position}, \mathit{Acid}, \mathit{Block})$ and $\texttt{helical}(\mathit{TypeHelical}, \mathit{Position}, \mathit{Acid}, \mathit{Acid}, \mathit{Block})$. Being in a *Block* of consecutive helical (resp. single) structure elements, the model will remain in the *Block* or transition to a $\texttt{single}$ element. The transition to a single (resp. helical) element only occurs at *Position* $\texttt{n}(\texttt{0})$. At all other positions $\texttt{n}(\mathit{Position})$, there were transitions from helical (resp. single) structure elements to helical (resp. single) structure elements at *Position* capturing the dynamics of the nucleotide pairs (resp. nucleotides) within structure elements. For instance,

66 signal structures and is very close to our data set. On a larger data set (with 400 structures) Horváth et al. report an error rate of 3.8% .

4. $\texttt{n}^m(\texttt{0})$ is shorthand for the recursive application of the functor $\texttt{n}$ on $\texttt{0}$ m times, i.e., for position m.

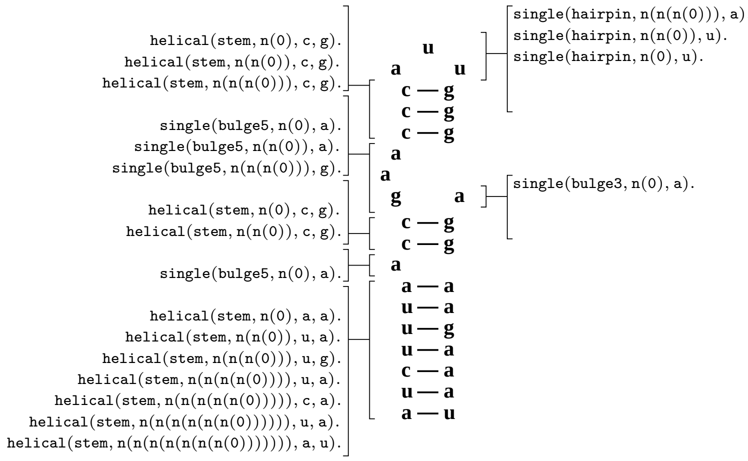

helical(stem, n(0), c, g).
helical(stem, n(n(0)), c, g).
helical(stem, n(n(n(0))), c, g).
single(bulge5, n(0), a).
single(bulge5, n(n(0)), a).
single(bulge5, n(n(n(0))), g).
helical(stem, n(0), c, g).
helical(stem, n(n(0)), c, g).
single(bulge5, n(0), a).
helical(stem, n(0), a, a).
helical(stem, n(n(0)), u, a).
helical(stem, n(n(n(0))), u, g).
helical(stem, n(n(n(n(0)))), u, a).
helical(stem, n(n(n(n(n(0))))), c, a).
helical(stem, n(n(n(n(n(n(0)))))), u, a).
helical(stem, n(n(n(n(n(n(n(0))))))), a, u).
single(hairpin, n(n(n(0))), a).
single(hairpin, n(n(0)), u).
single(hairpin, n(0), u).
single(bulge3, n(0), a).
u
a u
c — g
c — g
c — g
a
a
g a
c — g
c — g
a
a — a
u — a
u — g
u — a
c — a
u — a
a — u

Figure 5: The *chain* representation of a SECIS signal structure. The ground atoms are ordered clockwise starting with $\texttt{helical(stem, n(n(n(n(n(n(n(0))))))), a, u)}$ at the lower left-hand side corner.

the transitions for block $\texttt{n(0)}$ at position $\texttt{n(n(0))}$ were

$$
\begin{array}{lrcl}
a: & \texttt{he(stem, n(0), X, Y, n(0))} & \xleftarrow{p_a:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem, n(n(0)), X, Y, n(0)))} \\
b: & \texttt{he(stem, n(0), Y, X, n(0))} & \xleftarrow{p_b:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem, n(n(0)), X, Y, n(0)))} \\
c: & \texttt{he(stem, n(0), X, \_, n(0))} & \xleftarrow{p_c:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem, n(n(0)), X, Y, n(0)))} \\
d: & \texttt{he(stem, n(0), \_, Y, n(0))} & \xleftarrow{p_d:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem, n(n(0)), X, Y, n(0)))} \\
e: & \texttt{he(stem, n(0), \_, \_, n(0))} & \xleftarrow{p_e:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem, n(n(0)), X, Y, n(0)))}
\end{array}
$$

In total, there were 5 possible blocks as this was the maximal number of blocks of consecutive helical structure elements observed in the data. Overall, the LOHMM has 702 parameters. In contrast, the corresponding HMM has more than 16600 transitions validating **(B1)**.

Results: The LOO test log-likelihood was −63.7, and an EM iteration took on average 26 seconds.

Without the unification-based transitions *b*-*d*, i.e., using only the abstract transitions

$$
\begin{array}{lrcl}
a: & \texttt{he(stem, n(0), X, Y, n(0))} & \xleftarrow{p_a:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem, n(n(0)), X, Y, n(0)))} \\
e: & \texttt{he(stem, n(0), \_, \_, n(0))} & \xleftarrow{p_e:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem, n(n(0)), X, Y, n(0)))},
\end{array}
$$

the model has 506 parameters. The LOO test log-likelihood was −64.21, and an EM iteration took on average 20 seconds. The difference in LOO test log-likelihood is statistically significant (paired t-test, $p = 0.01$).

Omitting even transition a, the LOO test log-likelihood dropped to −66.06, and the average time per EM iteration was 18 seconds. The model has 341 parameters. The difference in average LOO log-likelihood is statistically significant (paired t-test, $p = 0.001$).

The results clearly show that unification can yield better LOO test log-likelihoods, i.e., **(B2)** holds.

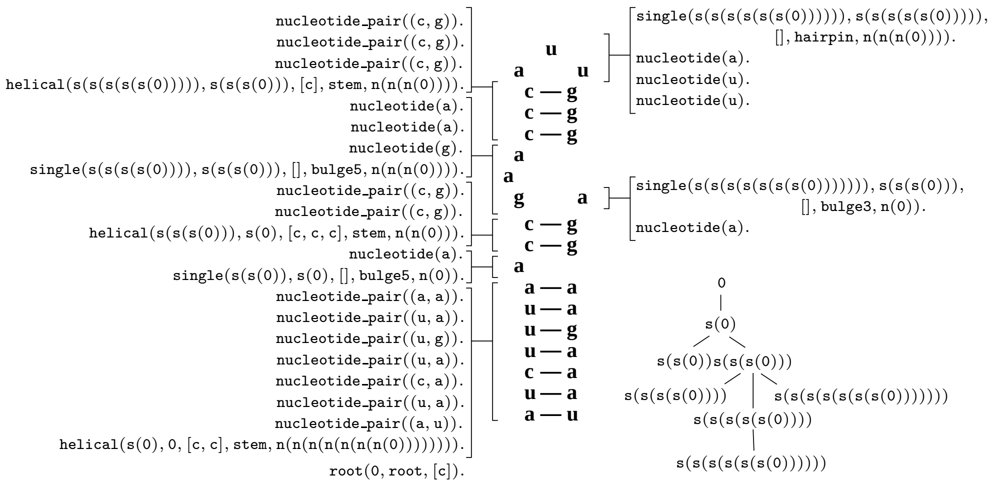


Figure 6: The *tree* representation of a SECIS signal structure. **(a)** The logical sequence, i.e., the sequence of ground atoms representing the SECIS signal structure. The ground atoms are ordered clockwise starting with root(0, root, [c]) in the lower left-hand side corner. **(b)** The tree formed by the secondary structure elements.

Tree Representation: In the *tree* representation (see Figure 6 **(a)**), the idea is to capture the tree structure formed by the secondary structure elements, see Figure 6 **(b)**. Each training instance is described as a sequence of ground facts over

$$\begin{array}{l}\texttt{root}(\texttt{0}, \texttt{root}, \#\mathit{Children}),\\ \texttt{helical}(\mathit{ID}, \mathit{ParentID}, \#\mathit{Children}, \mathit{Type}, \mathit{Size}),\\ \texttt{nucleotide\_pair}(\mathit{BasePair}),\\ \texttt{single}(\mathit{ID}, \mathit{ParentID}, \#\mathit{Children}, \mathit{Type}, \mathit{Size}),\\ \texttt{nucleotide}(\mathit{Base})\ .\end{array}$$

Here, *ID* and *ParentID* are natural numbers $\texttt{0}, \texttt{s(0)}, \texttt{s(s(0))}, \ldots$ encoding the child-parent relation, $\#$*Children* denotes the number[5] of children $[], [\texttt{c}], [\texttt{c}, \texttt{c}], \ldots$, *Type* is the type of the structure element such as $\texttt{stem}, \texttt{hairpin}, \ldots$, and *Size* is a natural number $\texttt{0}, \texttt{n(0)}, \texttt{n(n(0))}, \ldots$ Atoms $\texttt{root}(\texttt{0}, \texttt{root}, \#\mathit{Children})$ are used to root the topology. The maximal $\#$*Children* was 9 and the maximal *Size* was 13 as this was the maximal value observed in the data.

As trees can not easily be handled using HMMs, we used a LOHMM which basically encodes a PCFG. Due to Theorem 2, this is possible. The used LOHMM structure can be found in Appendix E. It processes the mRNA trees in in-order. Unification is only used for parsing the tree. As for the chain representation, we used a *Position* argument in the hidden states to encode the dynamics of nucleotides (nucleotide pairs) within secondary structure

5. Here, we use the Prolog short hand notation [·] for lists. A list either is the constant [] representing the empty list, or is a compound term with functor ./2 and two arguments, which are respectively the head and tail of the list. Thus [a, b, c] is the compound term .(a, .(b, .(c, []))).

elements. The maximal *Position* was again 13. In contrast to the chain representation, nucleotide pairs such as (a, u) are treated as constants. Thus, the argument *BasePair* consists of 16 elements.

Results: The LOO test log-likelihood was −55.56. Thus, exploiting the tree structure yields better probabilistic models. On average, an EM iteration took 14 seconds. Overall, the result shows that **(B4)** holds.

Although the Baum-Welch algorithm attempts to maximize a different objective function, namely the likelihood of the data, it is interesting to compare LOHMMs and RIBL in terms of classification accuracy.

Classification Accuracy: On the *chain* representation, the LOO accuracies of all LOHMMs were 99% (92/93). This is a considerable improvement on RIBL's 77.2% (51/66) LOO accuracy for this representation. On the *tree* representation, the LOHMM also achieved a LOO accuracy of 99% (92/93). This is comparable to RIBL's LOO accuracy of 97% (64/66) on this kind of representation.

Thus, already the chain LOHMMs show marked increases in LOO accuracy when compared to RIBL (Horváth et al., 2001). In order to achieve similar LOO accuracies, Horváth et al. (2001) had to make the tree structure available to RIBL as background knowledge. For LOHMMs, this had a significant influence on the LOO test log-likelihood, but not on the LOO accuracies. This clearly supports **(B3)**. Moreover, according to Horváth et al., the mRNA application can also be considered a success in terms of the application domain, although this was not the primary goal of our experiments. There exist also alternative parameter estimation techniques and other models, such as covariance models (Eddy & Durbin, 1994) or pair hidden Markov models (Sakakibara, 2003), that might have been used as well as a basis for comparison. However, as LOHMMs employ (inductive) logic programming principles, it is appropriate to compare with other systems within this paradigm such as RIBL.

7. Related Work

LOHMMs combine two different research directions. On the one hand, they are related to several extensions of HMMs and probabilistic grammars. On the other hand, they are also related to the recent interest in combining inductive logic programming principles with probability theory (De Raedt & Kersting, 2003, 2004).

In the first type of approaches, the underlying idea is to *upgrade* HMMs and probabilistic grammars to represent more structured state spaces.

Hierarchical HMMs (Fine, Singer, & Tishby, 1998), factorial HMMs (Ghahramani & Jordan, 1997), and HMMs based on tree automata (Frasconi, Soda, & Vullo, 2002) decompose the state variables into smaller units. In hierarchical HMMs states themselves can be HMMs, in factorial HMMs they can be factored into k state variables which depend on one another only through the observation, and in tree based HMMs the represented probability distributions are defined over tree structures. The key difference with LOHMMs is that these approaches do not employ the logical concept of unification. Unification is essential because it allows us to introduce abstract transitions, which do not *consist* of more detailed states. As our experimental evidence shows, sharing information among abstract states by means of unification can lead to more accurate model estimation. The same holds for *relational Markov models* (RMMs) (Anderson, Domingos, & Weld, 2002) to which LOHMMs are most closely related. In RMMs, states can be of different types, with each type described by a different set of variables. The domain of each variable can be hierarchically structured. The main differences between LOHMMs and RMMs are that RMMs do not either support variable binding nor unification nor hidden states.

The equivalent of HMMs for context-free languages are *probabilistic context-free grammars* (PCFGs). Like HMMs, they do not consider sequences of logical atoms and do not employ unification. Nevertheless, there is a formal resemblance between the Baum-Welch algorithms for LOHMMs and for PCFGs. In case that a LOHMM encodes a PCFG both algorithms are identical from a theoretical point of view. They re-estimate the parameters as the ratio of the expected number of times a transition (resp. production) is used and the expected number of times a transition (resp. production) might have been used. The proof of Theorem 2 assumes that the PCFG is given in Greibach normal form[6] (GNF) and uses a pushdown automaton to parse sentences. For grammars in GNF, pushdown automata are common for parsing. In contrast, the actual computations of the Baum-Welch algorithm for PCFGs, the so called Inside-Outside algorithm (Baker, 1979; Lari & Young, 1990), is usually formulated for grammars in *Chomsky normal form*[7]. The Inside-Outside algorithm can make use of the efficient CYK algorithm (Hopcroft & Ullman, 1979) for parsing strings.

An alternative to learning PCFGs from strings only is to learn from more structured data such as *skeletons*, which are derivation trees with the nonterminal nodes removed (Levy & Joshi, 1978). Skeletons are exactly the set of trees accepted by *skeletal tree automata* (STA). Informally, an STA, when given a tree as input, processes the tree bottom up, assigning a state to each node based on the states of that node's children. The STA accepts a tree iff it assigns a final state to the root of the tree. Due to this automata-based characterization of the skeletons of derivation trees, the learning problem of (P)CFGs can be reduced to the problem of an STA. In particular, STA techniques have been adapted to learning tree grammars and (P)CFGs (Sakakibara, 1992; Sakakibara et al., 1994) efficiently.

PCFGs have been extended in several ways. Most closely related to LOHMMs are *unification-based grammars* which have been extensively studied in computational linguistics. Examples are (stochastic) attribute-value grammars (Abney, 1997), probabilistic feature grammars (Goodman, 1997), head-driven phrase structure grammars (Pollard & Sag, 1994), and lexical-functional grammars (Bresnan, 2001). For learning within such frameworks, methods from undirected graphical models are used; see the work of Johnson (2003) for a description of some recent work. The key difference to LOHMMs is that only nonterminals are replaced with structured, more complex entities. Thus, observation sequences of flat symbols and not of atoms are modelled. Goodman's *probabilistic feature grammars* are an exception. They treat terminals and nonterminals as vectors of features. No abstraction is made, i.e., the feature vectors are ground instances, and no unification can be employed.

6. A grammar is in GNF iff all productions are of the form $\mathtt{A} \leftarrow \mathtt{a}V$ where $\mathtt{A}$ is a variable, $\mathtt{a}$ is exactly one terminal and V is a string of none or more variables.
7. A grammar is in CNF iff every production is of the form $\mathtt{A} \leftarrow \mathtt{B}, \mathtt{C}$ or $\mathtt{A} \leftarrow \mathtt{a}$ where $\mathtt{A}, \mathtt{B}$ and $\mathtt{C}$ are variables, and $\mathtt{a}$ is a terminal.

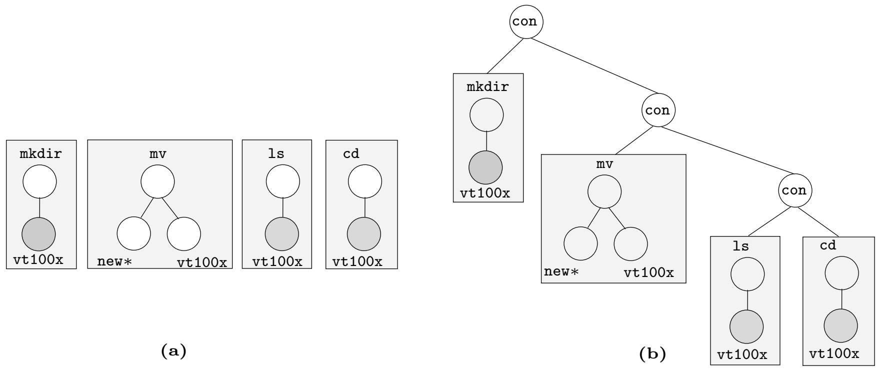


Figure 7: **(a)** Each atom in the logical sequence `mkdir(vt100x)`, `mv(new*, vt100x)`, `ls(vt100x)`, `cd(vt100x)` forms a tree. The shaded nodes denote shared labels among the trees. **(b)** The same sequence represented as a single tree. The predicate `con/2` represents the concatenation operator.

Therefore, the number of parameters that needs to be estimated becomes easily very large, data sparsity is a serious problem. Goodman applied smoothing to overcome the problem.

LOHMMs are generally related to (stochastic) tree automata (see e.g., Carrasco, Oncina, and Calera-Rubio, 2001). Reconsider the Unix command sequence `mkdir(vt100x), mv(new*, vt100x), ls(vt100x), cd(vt100x)` . Each atom forms a tree, see Figure 7 **(a)**, and, indeed, the whole sequence of atoms also forms a (degenerated) tree, see Figure 7 **(b)**. Tree automata process single trees vertically, e.g., bottom-up. A state in the automaton is assigned to every node in the tree. The state depends on the node label and on the states associated to the siblings of the node. They do not focus on sequential domains. In contrast, LOHMMs are intended for learning in sequential domains. They process sequences of trees horizontally, i.e., from left to right. Furthermore, unification is used to share information between consecutive sequence elements. As Figure 7 **(b)** illustrates, tree automata can only employ this information when allowing higher-order transitions, i.e., states depend on their node labels and on the states associated to predecessors $1, 2, \ldots$ levels down the tree.

In the second type of approaches, most attention has been devoted to developing highly expressive formalisms, such as e.g. PCUP (Eisele, 1994), PCLP (Riezler, 1998), SLPs (Muggleton, 1996), PLPs (Ngo & Haddawy, 1997), RBNs (Jaeger, 1997), PRMs (Friedman, Getoor, Koller, & Pfeffer, 1999), PRISM (Sato & Kameya, 2001), BLPs (Kersting & De Raedt, 2001b, 2001a), and DPRMs (Sanghai, Domingos, & Weld, 2003). LOHMMs can be seen as an attempt towards *downgrading* such highly expressive frameworks. Indeed, applying the main idea underlying LOHMMs to non-regular probabilistic grammar, i.e., replacing flat symbols with atoms, yields – in principle – stochastic logic programs (Muggleton, 1996). As a consequence, LOHMMs represent an interesting position on the expressiveness scale. Whereas they retain most of the essential logical features of the more expressive formalisms, they seem easier to understand, adapt and learn. This is akin to many contemporary considerations in *inductive logic programming* (Muggleton & De Raedt, 1994) and multi-relational data mining (Džeroski & Lavrač, 2001).

8. Conclusions

Logical hidden Markov models, a new formalism for representing probability distributions over sequences of logical atoms, have been introduced and solutions to the three central inference problems (evaluation, most likely state sequence and parameter estimation) have been provided. Experiments have demonstrated that unification can improve generalization accuracy, that the number of parameters of a LOHMM can be an order of magnitude smaller than the number of parameters of the corresponding HMM, that the solutions presented perform well in practice and also that LOHMMs possess several advantages over traditional HMMs for applications involving structured sequences.

Acknowledgments The authors thank Andreas Karwath and Johannes Horstmann for interesting collaborations on the protein data; Ingo Thon for interesting collaboration on analyzing the Unix command sequences; and Saul Greenberg for providing the Unix command sequence data. The authors would also like to thank the anonymous reviewers for comments which considerably improved the paper. This research was partly supported by the European Union IST programme under contract numbers IST-2001-33053 and FP6-508861 (Application of Probabilistic Inductive Logic Programming (APrIL) I and II). Tapani Raiko was supported by a Marie Curie fellowship at DAISY, HPMT-CT-2001-00251.

Appendix A. Proof of Theorem 1

Let $M = (\Sigma, \mu, \Delta, \Upsilon)$ be a LOHMM. To show that M specifies a time discrete stochastic process, i.e., a sequence of random variables $\langle X_t \rangle_{t=1,2,\ldots}$, where the domains of the random variable X_t is $\mathrm{hb}(\Sigma)$, the Herbrand base over Σ, we define the *immediate state operator* T_M-operator and the *current emission operator* E_M-operator.

Definition 4 *(T_M-Operator, E_M-Operator) The operators $T_M : 2^{\mathrm{hb}_\Sigma} \to 2^{\mathrm{hb}_\Sigma}$ and $E_M : 2^{\mathrm{hb}_\Sigma} \to 2^{\mathrm{hb}_\Sigma}$ are*

$$T_M(I) = \{H\sigma_B\sigma_H \mid \exists (p : H \xleftarrow{O} B) \in M : B\sigma_B \in I, H\sigma_B\sigma_H \in G_\Sigma(H)\}$$

$$\begin{aligned} E_M(I) = \{O\sigma_B\sigma_H\sigma_O \mid \exists (p : H \xleftarrow{O} B) \in M : B\sigma_B \in I,\ H\sigma_B\sigma_G \in G_\Sigma(H) \\ \text{and } O\sigma_B\sigma_H\sigma_O \in G_\Sigma(O)\} \end{aligned}$$

For each $i = 1, 2, 3, \ldots$, the set $T_M^{i+1}(\{\mathtt{start}\}) := T_M(T_M^i(\{\mathtt{start}\}))$ with $T_M^1(\{\mathtt{start}\}) := T_M(\{\mathtt{start}\})$ specifies the state set at clock i which forms a random variable Y_i. The set $U_M^i(\{\mathtt{start}\})$ specifies the possible symbols emitted when transitioning from i to $i+1$. It forms the variable U_i. Each Y_i (resp. U_i) can be extended to a random variable Z_i (resp. U_i) over hb_Σ:

$$P(Z_i = z) = \begin{cases} 0.0 & : \ z \notin T_M^i(\{\mathtt{start}\}) \\ P(Y_i = z) & : \ \text{otherwise} \end{cases}$$

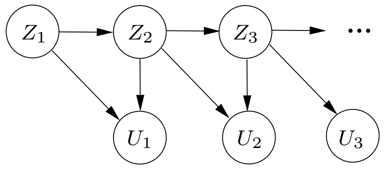


Figure 8: Discrete time stochastic process induced by a LOHMM. The nodes Z_i and U_i represent random variables over hb_Σ.

Figure 8 depicts the influence relation among Z_i and U_i. Using standard arguments from probability theory and noting that

$$P(U_i = U_i \mid Z_{i+1} = z_{i+1}, Z_i = z_i) = \frac{P(Z_{i+1} = z_{i+1}, U_i = u_i \mid Z_i)}{\sum_{u_i} P(Z_{i+1}, u_i \mid Z_i)}$$

$$\text{and } P(Z_{i+1} \mid Z_i) = \sum_{u_i} P(Z_{i+1}, u_i \mid Z_i)$$

where the probability distributions are due to equation (1), it is easy to show that Kolmogorov's extension theorem (see Bauer, 1991; Fristedt and Gray, 1997) holds. Thus, M specifies a unique probability distribution over $\bigotimes_{i=1}^{t}(Z_i \times U_i)$ for each $t > 0$ and in the limit $t \to \infty$. □

Appendix B. Moore Representations of LOHMMs

For HMMs, *Moore* representations, i.e., output symbols are associated with states and *Mealy* representations, i.e., output symbols are associated with transitions, are equivalent. In this appendix, we will investigate to which extend this also holds for LOHMMs.

Let L be a Mealy-LOHMM according to definition 3. In the following, we will derive the notation of an equivalent LOHMM L' in Moore representation where there are abstract transitions and abstract emissions (see below). Each predicate $\mathtt{b}/n$ in L is extended to $\mathtt{b}/n+1$ in L'. The domains of the first n arguments are the same as for $\mathtt{b}/n$. The last argument will store the observation to be emitted. More precisely, for each abstract transition

$$p : \mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l}) \xleftarrow{\mathtt{o}(\mathtt{v_1}, \ldots, \mathtt{v_k})} \mathtt{b}(\mathtt{u_1}, \ldots, \mathtt{u_n})$$

in L, there is an abstract transition

$$p : \mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l}, \mathtt{o}(\mathtt{v'_1}, \ldots, \mathtt{v'_k})) \leftarrow \mathtt{b}(\mathtt{u_1}, \ldots, \mathtt{u_n}, \_)$$

in L'. The primes in $\mathtt{o}(\mathtt{v'_1}, \ldots, \mathtt{v'_k})$ denote that we replaced each free [8] variables $\mathtt{o}(\mathtt{v_1}, \ldots, \mathtt{v_k})$ by some distinguished constant symbol, say #. Due to this, it holds that

$$\mu(\mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l})) = \mu(\mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l}, \mathtt{o}(\mathtt{v'_1}, \ldots, \mathtt{v'_k}))) \, , \tag{6}$$

8. A variable $\mathtt{X} \in \mathrm{vars}(\mathtt{o}(\mathtt{v_1}, \ldots, \mathtt{v_k}))$ is free iff $\mathtt{X} \notin \mathrm{vars}(\mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l})) \cup \mathrm{vars}(\mathtt{b}(\mathtt{u_1}, \ldots, \mathtt{u_n}))$.

and L''s output distribution can be specified using *abstract emissions* which are expressions of the form

$$1.0 : \mathtt{o}(\mathtt{v_1}, \ldots, \mathtt{v_k}) \leftarrow \mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l}, \mathtt{o}(\mathtt{v}'_1, \ldots, \mathtt{v}'_k)) \ . \tag{7}$$

The semantics of an abstract transition in L' is that being in some state $\mathtt{S}'_t \in G_{\Sigma'}(\mathtt{b}(\mathtt{u_1}, \ldots, \mathtt{u_n}, \_))$ the system will make a transition into state $\mathtt{S}'_{t+1} \in G_{\Sigma'}(\mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l}, \mathtt{o}(\mathtt{v}'_1, \ldots, \mathtt{v}'_k)))$ with probability

$$p \cdot \mu(\mathtt{S}'_{t+1} \mid \mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l}, \mathtt{o}(\mathtt{v}'_1, \ldots, \mathtt{v}'_k)) \mid \sigma_{\mathtt{S}'_t}) \tag{8}$$

where $\sigma_{\mathtt{S}'_t} = \mathrm{mgu}(\mathtt{S}'_t, \mathtt{b}(\mathtt{u_1}, \ldots, \mathtt{u_n}, \_))$. Due to Equation (6), Equation (8) can be rewritten as

$$p \cdot \mu(\mathtt{S}'_{t+1} \mid \mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l}) \mid \sigma_{\mathtt{S}'_t}) \ .$$

Due to equation (7), the system will emit the output symbol $\mathtt{o}_{t+1} \in G_{\Sigma'}(\mathtt{o}(\mathtt{v_1}, \ldots, \mathtt{v_k}))$ in state $\mathtt{S}'_{t+1}$ with probability

$$\mu(\mathtt{o}_{t+1} \mid \mathtt{o}(\mathtt{v_1}, \ldots, \mathtt{v_k})\sigma_{\mathtt{S}'_{t+1}}\sigma_{\mathtt{S}'_t})$$

where $\sigma_{\mathtt{S}'_{t+1}} = \mathrm{mgu}(\mathtt{h}(\mathtt{w_1}, \ldots, \mathtt{w_l}, \mathtt{o}(\mathtt{v}'_1, \ldots, \mathtt{v}'_k)), \mathtt{S}'_{t+1})$. Due to the construction of L', there exists a triple $(\mathtt{S}_t, \mathtt{S}_{t+1}, \mathtt{O_{t+1}})$ in L for each triple $(\mathtt{S}'_t, \mathtt{S}'_{t+1}, \mathtt{O}_{t+1})$, $t > 0$, in L' (and vise versa). Hence,both LOHMMs assign the same overall transition probability.

L and L' differ only in the way the initialize sequences $\langle(\mathtt{S}'_t, \mathtt{S}'_{t+1}, \mathtt{O}_{t+1}\rangle_{t=0,2\ldots,T}$ (resp. $\langle(\mathtt{S}_t, \mathtt{S}_{t+1}, \mathtt{O_{t+1}}\rangle_{t=0,2\ldots,T}$). Whereas L starts in some state $\mathtt{S_0}$ and makes a transition to $\mathtt{S_1}$ emitting $\mathtt{O_1}$, the Moore-LOHMM L' is supposed to emit a symbol $\mathtt{O_0}$ in $\mathtt{S}'_0$ before making a transition to $\mathtt{S}'_1$. We compensate for this using the prior distribution. The existence of the correct prior distribution for L' can be seen as follows. In L, there are only finitely many states reachable at time $t = 1$, i.e, $P_L(q_0 = \mathtt{S}) > 0$ holds for only a finite set of ground states $\mathtt{S}$. The probability $P_L(q_0 = \mathtt{s})$ can be computed similar to $\alpha_1(\mathtt{S})$. We set $t = 1$ in line 6, neglecting the condition on $\mathtt{O}_{t-1}$ in line 10, and dropping $\mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})$ from line 14. Completely listing all states $\mathtt{S} \in S_1$ together with $P_L(q_0 = \mathtt{S})$, i.e., $P_L(q_0 = \mathtt{S}) : \mathtt{S} \leftarrow \mathtt{start}$, constitutes the prior distribution of L'.

The argumentation basically followed the approach to transform a Mealy machine into a Moore machine (see e.g., Hopcroft and Ullman, 1979). Furthermore, the mapping of a Moore-LOHMM – as introduced in the present section – into a Mealy-LOHMM is straightforward.

Appendix C. Proof of Theorem 2

Let T be a terminal alphabet and N a nonterminal alphabet. A *probabilistic context-free grammar* (PCFG) G consists of a distinguished start symbol $S \in N$ plus a finite set of productions of the form $p : X \to \alpha$, where $X \in N$, $\alpha \in (N \cup T)^*$ and $p \in [0, 1]$. For all $X \in N$, $\sum_{:X\to\alpha} p = 1$. A PCFG defines a stochastic process with sentential forms as states, and leftmost rewriting steps as transitions. We denote a single rewriting operation of the grammar by a single arrow $\to$. If as a result of one ore more rewriting operations we are able to rewrite $\beta \in (N \cup T)^*$ as a sequence $\gamma \in (N \cup T)^*$ of nonterminals and terminals, then we write $\beta \Rightarrow^* \gamma$. The probability of this rewriting is the product of all probability values associated to productions used in the derivation. We assume G to be consistent, i.e., that the sum of all probabilities of derivations $S \Rightarrow^* \beta$ such that $\beta \in T^*$ sum to 1.0.

We can assume that the PCFG G is in Greibach normal form. This follows from Abney et al.'s (1999) Theorem 6 because G is consistent. Thus, every production $P \in G$ is of the form $p : X \rightarrow aY_1 \ldots Y_n$ for some $n \geq 0$. In order to encode G as a LOHMM M, we introduce (1) for each non-terminal symbol X in G a constant symbol nX and (2) for each terminal symbol t in G a constant symbol t. For each production $P \in G$, we include an abstract transition of the form $p : \mathtt{stack}([\mathtt{nY_1}, \ldots, \mathtt{nY_n}|\mathtt{S}]) \overset{\mathtt{a}}{\leftarrow} \mathtt{stack}([\mathtt{nX}|\mathtt{S}])$, if $n > 0$, and $p : \mathtt{stack}(\mathtt{S}) \overset{\mathtt{a}}{\leftarrow} \mathtt{stack}([\mathtt{nX}|\mathtt{S}])$, if $n = 0$. Furthermore, we include $1.0 : \mathtt{stack}([\mathtt{s}]) \leftarrow \mathtt{start}$ and $1.0 : \mathtt{end} \xleftarrow{\mathtt{end}} \mathtt{stack}([])$. It is now straightforward to prove by induction that M and G are equivalent. □

Appendix D. Logical Hidden Markov Model for Unix Command Sequences

The LOHMMs described below model Unix command sequences triggered by mkdir. To this aim, we transformed the original Greenberg data into a sequence of logical atoms over com, mkdir(Dir, LastCom), ls(Dir, LastCom), cd(Dir, Dir, LastCom), cp(Dir, Dir, LastCom) and mv(Dir, Dir, LastCom). The domain of LastCom was {start, com, mkdir, ls, cd, cp, mv}. The domain of Dir consisted of all argument entries for mkdir, ls, cd, cp, mv in the original dataset. Switches, pipes, etc. were neglected, and paths were made absolute. This yields 212 constants in the domain of Dir. All original commands, which were not mkdir, ls, cd, cp, or mv, were represented as com. If mkdir did not appear within 10 time steps before a command $C \in$ {ls, cd, cp, mv}, C was represented as com. Overall, this yields more than 451000 ground states that have to be covered by a Markov model.

The "unification" LOHMM U basically implements a second order Markov model, i.e., the probability of making a transition depends upon the current state and the previous state. It has 542 parameters and the following structure:

```
               com ← start.               com ← com.
 mkdir(Dir, start) ← start.   mkdir(Dir, com) ← com.
                                          end ← com.
```

Furthermore, for each $C \in$ {start, com} there are

```
mkdir(Dir, com) ← mkdir(Dir,C).        cd(_, mkdir) ← mkdir(Dir,C).
  mkdir(_, com) ← mkdir(Dir,C).   cp(_, Dir, mkdir) ← mkdir(Dir,C).
            com ← mkdir(Dir,C).   cp(Dir, _, mkdir) ← mkdir(Dir,C).
            end ← mkdir(Dir,C).     cp(_, _, mkdir) ← mkdir(Dir,C).
 ls(Dir, mkdir) ← mkdir(Dir,C).   mv(_, Dir, mkdir) ← mkdir(Dir,C).
   ls(_, mkdir) ← mkdir(Dir,C).   mv(Dir, _, mkdir) ← mkdir(Dir,C).
 cd(Dir, mkdir) ← mkdir(Dir,C).     mv(_, _, mkdir) ← mkdir(Dir,C).
```

together with for each $C \in \{\mathtt{mkdir}, \mathtt{ls}, \mathtt{cd}, \mathtt{cp}, \mathtt{mv}\}$ and for each $C_1 \in \{\mathtt{cd}, \mathtt{ls}\}$ (resp. $C_2 \in \{\mathtt{cp}, \mathtt{mv}\}$)

$$\begin{array}{rcl rcl}
\mathtt{mkdir(Dir,com)} & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{mkdir(\_,com)} & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{mkdir(\_,com)} & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{com} & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{com} & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{end} & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{end} & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{ls(From},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{ls(Dir},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{ls(To},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{ls(\_},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{ls(\_},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{cd(Dir},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{cd(From},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{cd(\_},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{cd(To},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{cp(\_,Dir},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{cd(\_},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{cp(Dir,\_},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{cp(From,\_},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{cp(\_,\_},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{cp(\_,To},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{mv(\_,Dir},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{cp(\_,\_},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{mv(Dir,\_},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{mv(From,\_},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
\mathtt{mv(\_,\_},C_1) & \leftarrow & C_1(\mathtt{Dir},C). & \mathtt{mv(\_,To},C_2) & \leftarrow & C_2(\mathtt{From,To},C). \\
 & & & \mathtt{mv(\_,\_},C_2) & \leftarrow & C_2(\mathtt{From,To},C).
\end{array}$$

Because all states are fully observable, we omitted the output symbols associated with clauses, and, for the sake of simplicity, we omitted associated probability values.

The "no unification" LOHMM N is the variant of U where no variables were shared such as

$$\begin{array}{rcl rcl}
\mathtt{mkdir(\_,com)} & \leftarrow & \mathtt{cp(From,To},C). & \mathtt{ls(\_,cp)} & \leftarrow & \mathtt{cp(From,To},C). \\
\mathtt{com} & \leftarrow & \mathtt{cp(From,To},C). & \mathtt{cd(\_,cp)} & \leftarrow & \mathtt{cp(From,To},C). \\
\mathtt{end} & \leftarrow & \mathtt{cp(From,To},C). & \mathtt{cp(\_,\_,cp)} & \leftarrow & \mathtt{cp(From,To},C). \\
 & & & \mathtt{mv(\_,\_,cp)} & \leftarrow & \mathtt{cp(From,To},C).
\end{array}$$

Because only transitions are affected, N has 164 parameters less than U, i.e., 378.

Appendix E. Tree-based LOHMM for mRNA Sequences

The LOHMM processes the nodes of mRNA trees in in-order. The structure of the LOHMM is shown at the end of the section. There are copies of the shaded parts. Terms are abbreviated using their starting alphanumerical; `tr` stands for `tree`, `he` for `helical`, `si` for `single`, `nuc` for `nucleotide`, and `nuc_p` for `nucleotide_pair`.

The domain of *#Children* covers the maximal branching factor found in the data, i.e., $\{[\mathtt{c}], [\mathtt{c},\mathtt{c}], \ldots, [\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c},\mathtt{c}]\}$; the domain of *Type* consists of all types occurring in the data, i.e., $\{\mathtt{stem}, \mathtt{single}, \mathtt{bulge3}, \mathtt{bulge5}, \mathtt{hairpin}\}$; and for *Size*, the domain covers the maximal length of a secondary structure element in the data, i.e., the longest sequence of consecutive bases respectively base pairs constituting a secondary structure element. The length was encoded as $\{\mathtt{n}^1(\mathtt{0}), \mathtt{n}^2(\mathtt{0}), \ldots, \mathtt{n}^{13}(\mathtt{0})\}$ where $\mathtt{n}^m(\mathtt{0})$ denotes the recursive application of the functor `n` m times. For *Base* and *BasePair*, the domains were the 4 bases respectively the 16 base pairs. In total, there are 491 parameters.

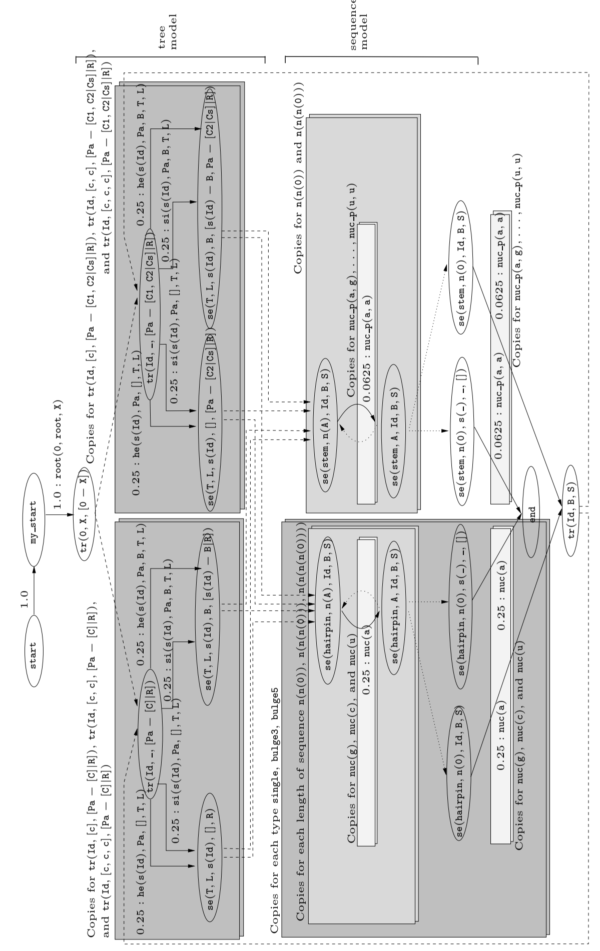

start
1.0
my_start
1.0 : root(0, root, X)
tr(0, X, [0 − X])
Copies for tr(Id, [c], [Pa − [C]|R]), tr(Id, [c, c], [Pa − [C]|R]), and tr(Id, [c, c, c], [Pa − [C]|R])
0.25 : he(s(Id), Pa, [], T, L)
tr(Id, _, [Pa − [C]|R])
0.25 : he(s(Id), Pa, B, T, L)
0.25 : si(s(Id), Pa, B, T, L)
0.25 : si(s(Id), Pa, [], T, L)
se(T, L, s(Id), [], R)
se(T, L, s(Id), B, [s(Id) − B|R])
Copies for tr(Id, [c], [Pa − [C1, C2|Cs]|R]), tr(Id, [c, c], [Pa − [C1, C2|Cs]|R]), and tr(Id, [c, c, c], [Pa − [C1, C2|Cs]|R])
0.25 : he(s(Id), Pa, [], T, L)
tr(Id, _, [Pa − [C1, C2|Cs]|R])
0.25 : he(s(Id), Pa, B, T, L)
0.25 : si(s(Id), Pa, B, T, L)
0.25 : si(s(Id), Pa, [], T, L)
se(T, L, s(Id), [], [Pa − [C2|Cs]|R])
se(T, L, s(Id), B, [s(Id) − B, Pa − [C2|Cs]|R])
tree model
Copies for each type single, bulge3, bulge5
Copies for each length of sequence n(n(0)), n(n(n(0))), n(n(n(n(0))))
se(hairpin, n(A), Id, B, S)
Copies for nuc(g), nuc(c), and nuc(u)
0.25 : nuc(a)
se(hairpin, A, Id, B, S)
Copies for n(n(0)) and n(n(n(0)))
se(stem, n(A), Id, B, S)
Copies for nuc_p(a, g), . . . , nuc_p(u, u)
0.0625 : nuc_p(a, a)
se(stem, A, Id, B, S)
sequence model
se(hairpin, n(0), Id, B, S)
se(hairpin, n(0), s(_), _, [])
se(stem, n(0), s(_), _, [])
se(stem, n(0), Id, B, S)
0.25 : nuc(a)
0.25 : nuc(a)
0.0625 : nuc_p(a, a)
0.0625 : nuc_p(a, a)
Copies for nuc(g), nuc(c), and nuc(u)
Copies for nuc_p(a, g), . . . , nuc_p(u, u)
end
tr(Id, B, S)

Figure 9: The mRNA LOHMM structure. The symbol _ denotes anonymous variables which are read and treated as distinct, new variables each time they are encountered. There are copies of the shaded part. Terms are abbreviated using their starting alphanumerical; `tr` stands for `tree`, `se` for `structure_element`, `he` for `helical`, `si` for `single`, `nuc` for `nucleotide`, and `nuc_p` for `nucleotide_pair`.

References

Abney, S. (1997). Stochastic Attribute-Value Grammars. *Computational Linguistics*, *23*(4), 597–618.

Abney, S., McAllester, D., & Pereira, F. (1999). Relating probabilistic grammars and automata. In *Proceedings of 37th Annual Meeting of the Association for Computational Linguistics (ACL-1999)*, pp. 542–549. Morgan Kaufmann.

Anderson, C., Domingos, P., & Weld, D. (2002). Relational Markov Models and their Application to Adaptive Web Navigation. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pp. 143–152 Edmonton, Canada. ACM Press.

Baker, J. (1979). Trainable grammars for speech recognition. In *Speech communication paper presented at th 97th Meeting of the Acoustical Society of America*, pp. 547–550 Boston, MA.

Bauer, H. (1991). *Wahrscheinlichkeitstheorie* (4. edition). Walter de Gruyter, Berlin, New York.

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, *3*, 1–8.

Bohnebeck, U., Horváth, T., & Wrobel, S. (1998). Term comparison in first-order similarity measures. In *Proceedings of the Eigth International Conference on Inductive Logic Programming (ILP-98)*, Vol. 1446 of *LNCS*, pp. 65–79. Springer.

Bresnan, J. (2001). *Lexical-Functional Syntax.* Blackwell, Malden, MA.

Carrasco, R., Oncina, J., & Calera-Rubio, J. (2001). Stochastic inference of regular tree languages. *Machine Learning*, *44*(1/2), 185–197.

Chandonia, J., Hon, G., Walker, N., Lo Conte, L., P.Koehl, & Brenner, S. (2004). The ASTRAL compendium in 2004. *Nucleic Acids Research*, *32*, D189–D192.

Davison, B., & Hirsh, H. (1998). Predicting Sequences of User Actions. In *Predicting the Future: AI Approaches to Time-Series Analysis*, pp. 5–12. AAAI Press.

De Raedt, L., & Kersting, K. (2003). Probabilistic Logic Learning. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, *5*(1), 31–48.

De Raedt, L., & Kersting, K. (2004). Probabilistic Inductive Logic Programming. In Ben-David, S., Case, J., & Maruoka, A. (Eds.), *Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT-2004)*, Vol. 3244 of *LNCS*, pp. 19–36 Padova, Italy. Springer.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press.

Džeroski, S., & Lavrač, N. (Eds.). (2001). *Relational data mining.* Springer-Verlag, Berlin.

Eddy, S., & Durbin, R. (1994). RNA sequence analysis using covariance models. *Nucleic Acids Res.*, *22*(11), 2079–2088.

Eisele, A. (1994). Towards probabilistic extensions of contraint-based grammars. In Dörne, J. (Ed.), *Computational Aspects of Constraint-Based Linguistics Decription-II*. DYNA-2 deliverable R1.2.B.

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical hidden markov model: analysis and applications. *Machine Learning*, *32*, 41–62.

Frasconi, P., Soda, G., & Vullo, A. (2002). Hidden markov models for text categorization in multi-page documents. *Journal of Intelligent Information Systems*, *18*, 195–217.

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. In *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-1999)*, pp. 1300–1307. Morgan Kaufmann.

Fristedt, B., & Gray, L. (1997). *A Modern Approach to Probability Theory*. Probability and its applications. Birkhäuser Boston.

Ghahramani, Z., & Jordan, M. (1997). Factorial hidden Markov models. *Machine Learning*, *29*, 245–273.

Goodman, J. (1997). Probabilistic feature grammars. In *Proceedings of the Fifth International Workshop on Parsing Technologies (IWPT-97)* Boston, MA, USA.

Greenberg, S. (1988). Using Unix: collected traces of 168 users. Tech. rep., Dept. of Computer Science, University of Calgary, Alberta.

Hopcroft, J., & Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company.

Horváth, T., Wrobel, S., & Bohnebeck, U. (2001). Relational Instance-Based learning with Lists and Terms. *Machine Learning*, *43*(1/2), 53–80.

Hubbard, T., Murzin, A., Brenner, S., & Chotia, C. (1997). *SCOP*: a structural classification of proteins database. *NAR*, *27*(1), 236–239.

Jacobs, N., & Blockeel, H. (2001). The Learning Shell: Automated Macro Construction. In *User Modeling 2001*, pp. 34–43.

Jaeger, M. (1997). Relational Bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 266–273. Morgan Kaufmann.

Katz, S. (1987). Estimation of probabilities from sparse data for hte language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing (ASSP)*, *35*, 400–401.

Kersting, K., & De Raedt, L. (2001a). Adaptive Bayesian Logic Programs. In Rouveirol, C., & Sebag, M. (Eds.), *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, Vol. 2157 of *LNAI*, pp. 118–131. Springer.

Kersting, K., & De Raedt, L. (2001b). Towards Combining Inductive Logic Programming with Bayesian Networks. In Rouveirol, C., & Sebag, M. (Eds.), *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, Vol. 2157 of *LNAI*, pp. 118–131. Springer.

Kersting, K., & Raiko, T. (2005). 'Say EM' for Selecting Probabilistic Models for Logical Sequences. In Bacchus, F., & Jaakkola, T. (Eds.), *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, UAI 2005*, pp. 300–307 Edinburgh, Scotland.

Kersting, K., Raiko, T., Kramer, S., & De Raedt, L. (2003). Towards discovering structural signatures of protein folds based on logical hidden markov models. In Altman, R., Dunker, A., Hunter, L., Jung, T., & Klein, T. (Eds.), *Proceedings of the Pacific Symposium on Biocomputing (PSB-03)*, pp. 192–203 Kauai, Hawaii, USA. World Scientific.

Koivisto, M., Kivioja, T., Mannila, H., Rastas, P., & Ukkonen, E. (2004). Hidden Markov Modelling Techniques for Haplotype Analysis. In Ben-David, S., Case, J., & Maruoka, A. (Eds.), *Proceedings of 15th International Conference on Algorithmic Learning Theory (ALT-04)*, Vol. 3244 of *LNCS*, pp. 37–52. Springer.

Koivisto, M., Perola, M., Varilo, T., Hennah, W., Ekelund, J., Lukk, M., Peltonen, L., Ukkonen, E., & Mannila, H. (2002). An MDL method for finding haplotype blocks and for estimating the strength of haplotype block boundaries. In Altman, R., Dunker, A., Hunter, L., Jung, T., & Klein, T. (Eds.), *Proceedings of the Pacific Symposium on Biocomputing (PSB-02)*, pp. 502–513. World Scientific.

Korvemaker, B., & Greiner, R. (2000). Predicting UNIX command files: Adjusting to user patterns. In *Adaptive User Interfaces: Papers from the 2000 AAAI Spring Symposium*, pp. 59–64.

Kulp, D., Haussler, D., Reese, M., & Eeckman, F. (1996). A Generalized Hidden Markov Model for the Recognition of Human Genes in DNA. In States, D., Agarwal, P., Gaasterland, T., Hunter, L., & Smith, R. (Eds.), *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology,(ISMB-96)*, pp. 134–142 St. Louis, MO, USA. AAAI.

Lane, T. (1999). Hidden Markov Models for Human/Computer Interface Modeling. In Rudström, Å. (Ed.), *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pp. 35–44 Stockholm, Sweden.

Lari, K., & Young, S. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, *4*, 35–56.

Levy, L., & Joshi, A. (1978). Skeletal structural descriptions. *Information and Control*, *2*(2), 192–211.

McLachlan, G., & Krishnan, T. (1997). *The EM Algorithm and Extensions.* Wiley, New York.

Mitchell, T. M. (1997). *Machine Learning*. The McGraw-Hill Companies, Inc.

Muggleton, S. (1996). Stochastic logic programs. In De Raedt, L. (Ed.), *Advances in Inductive Logic Programming*, pp. 254–264. IOS Press.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19*(20), 629–679.

Ngo, L., & Haddawy, P. (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, *171*, 147–177.

Pollard, C., & Sag, I. (1994). *Head-driven Phrase Structure Grammar*. The University of Chicago Press, Chicago.

Rabiner, L., & Juang, B. (1986). An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, *3*(1), 4–16.

Riezler, S. (1998). Statistical inference and probabilistic modelling for constraint-based nlp. In Schrder, B., Lenders, W., & und T. Portele, W. H. (Eds.), *Proceedings of the 4th Conference on Natural Language Processing (KONVENS-98)*. Also as CoRR cs.CL/9905010.

Sakakibara, Y. (1992). Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, *97*(1), 23–60.

Sakakibara, Y. (2003). Pair hidden markov models on tree structures. *Bioinformatics*, *19*(Suppl.1), i232–i240.

Sakakibara, Y., Brown, M., Hughey, R., Mian, I., Sjolander, K., & Underwood, R. (1994). Stochastic context-free grammars for tRNA modelling. *Nucleic Acids Research*, *22*(23), 5112–5120.

Sanghai, S., Domingos, P., & Weld, D. (2003). Dynamic probabilistic relational models. In Gottlob, G., & Walsh, T. (Eds.), *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 992–997 Acapulco, Mexico. Morgan Kaufmann.

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)*, *15*, 391–454.

Schölkopf, B., & Warmuth, M. (Eds.). (2003). *Learning and Parsing Stochastic Unification-Based Grammars*, Vol. 2777 of *LNCS*. Springer.

Turcotte, M., Muggleton, S., & Sternberg, M. (2001). The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Machine Learning*, *43*(1/2), 81–95.

Won, K., Prügel-Bennett, A., & Krogh, A. (2004). The Block Hidden Markov Model for Biological Sequence Analysis. In Negoita, M., Howlett, R., & Jain, L. (Eds.), *Proceedings of the Eighth International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES-04)*, Vol. 3213 of *LNCS*, pp. 64–70. Springer.

Publication 8

K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards Discovering Structural Signatures of Protein Folds based on Logical Hidden Markov Models. In the *Proceedings of the Pacific Symposium on Biocomputing (PSB-2003)*, pp. 192–203, Kauai, Hawaii, January 3–7, 2003.

Towards Discovering Structural Signatures of Protein Folds based on Logical Hidden Markov Models

Kristian Kersting[1], T. Raiko[1,2], S. Kramer[1], L. De Raedt[1]

[1] *Institute for Computer Science*
Machine Learning Lab
University of Freiburg
Georges-Koehler-Allee 079
79112 Freiburg, Germany

[2] *Helsinki University of Technology*
Laboratory of Computer and
Information Science,
P.O. Box 5400,
02015 HUT, Finland

With the growing number of determined protein structures and the availability of classification schemes, it becomes increasingly important to develop computer methods that automatically extract structural signatures for classes of proteins. In this paper, we introduce and apply a new Machine Learning technique, Logical Hidden Markov Models (LOHMMs), to the task of finding structural signatures of folds according to the classification scheme SCOP. Our results indicate that LOHMMs are applicable to this task and possess several advantages over other approaches.

1 Introduction

In recent years, the number of proteins with determined structure has been growing rapidly due to large-scale structural genomics projects. Consequently, the Protein Data Bank (PDB) is growing at high rates. In parallel, researchers have developed classification schemes of proteins based on their sequence, structure and function. The development of classification schemes is a common scientific activity to make sense and gain a deeper understanding of experimental data. Given the determined structures and classification schemes, the discovery of structural characteristics of protein classes becomes an important topic. The primary interest there is to gain insights into structural characteristics of fold classes, but ultimately structural signatures should also be useful for the prediction of protein folds. In fact, some successful approaches in the CASP predictive exercises made use of knowledge about structural signatures. So far, most signatures have been discovered by human experts based on extensive manual/visual inspection of the data. However, few experts in the world are in a position to find/provide these signatures, and few systematic attempts exist to catalog known signatures. So, there is a need to develop computer methods that automatically extract structural signatures in a systematic way [11].

Recently, Hidden Markov Models (HMM) have been used to analyze classes in SCOP [6]. HMMs are among the most widely and successfully used tools for the analysis of sequence data in bioinformatics. Despite their successes, however, it is well-known that HMMs have a number of weaknesses. One of the major weaknesses is that HMMs handle only flat sequences, i.e. sequences of unstructured symbols. In this paper we will overcome this weakness by introducing Logical Hidden Markov Models (LOHMMs).

This paper is organized as follows. In Section 2, we present the task and the dataset. In Section 3, we introduce LOHMMs. Section 4 describes experiments with LOHMMs for the discovery of structural signatures. Subsequently, we discuss related work and conclude.

2 Task and Dataset

In this section, we describe the task of finding structural signatures of protein folds and the dataset used. The basis of our study is the SCOP (Structural Classification of Proteins) database due to A. Murzin and maintained by the MRC Laboratory of Molecular Biology. Our goal was to find structural characteristics of the domains at the second level of the SCOP hierarchy, i.e., the level of folds. In our study, we focused on alpha and beta proteins (a/b), a class consisting of domains with mainly parallel beta sheets (beta-alpha-beta units). From this class, we chose the five most populated subclasses, that is, folds: TIM beta/alpha-barrel, NAD(P)-binding Rossmann-fold domains, Ribosomal protein L4, glucosamine 6-phosphate deaminase/isomerase and and leucine aminopeptidas. The overall set-up is quite similar to the one in [12,11]. The data have been extracted automatically from the PDB release #96 and SCOP version 1.57.

Information for domains from the above five folds was extracted in the form of "logical sequences" of secondary structure elements. Logical sequences are sequences of logical atoms. An example of such a sequence (corresponding to a Ribosomal protein L4) is:

$$st(null, 2), he(h(right, alpha), 6), st(plus, 2), he(h(right, alpha), 4), st(plus, 2),$$
$$he(h(right, alpha), 4), st(plus, 3), he(h(right, alpha), 4),$$
$$st(plus, 1), he(h(right, alpha), 6).$$

There are two predicates *he* and *st*. Atoms $he(Type, Length)$ model helices of a certain type and length, whereas atoms $st(Orientation, Length)$ model strands of a certain orientation and length. The helix types are: $h(left, alpha), h(right, alpha), h(left, gamma), h(right, gamma), h(left, omega), h(right, omega), h(right, pi), h(right, 3to10)$, *27ribbon* and *polyproline*. The orientation of strands can be *null* (the beginning of a sheet), *plus* (a parallel strand of a sheet), or *minus* (an anti-parallel strand of

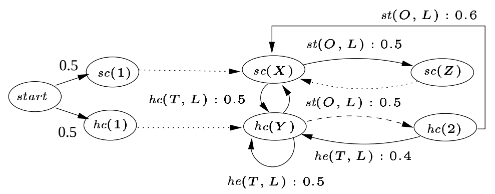


Figure 1: A logical hidden Markov model encoding default reasoning. The dashed edge represents a *more general than* relation.

a sheet). The length is defined as the number of acids and was quantized in the experiments (see below).

For each of the above five folds, we are modelling their domains in terms of their secondary structure using a logical variant of HMMs. So, what can be expected from an application of LOHMMs to this task? First of all, it should be clear that we do not obtain structural signatures for each of these classes immediately. What we obtain instead, is a model for each fold. Each model provides a precise probabilistic and logical characterization of the respective fold. Structural signatures can then be found by a comparison of models. As will be shown below, it is quite easy to find characteristics upon inspection of the trained models.

3 Logical (Hidden) Markov Models

Logical (hidden) Markov models (LOHMM) extend the unstructured model representation of HMMs [10,9] by incorporating complex, internal structure into the specification of transitions (and therefore of emissions) between states.

Sets of states are summarized by *abstract states*, which are represented by logical atoms. A logical atom then represents all states that can be obtained by instantiating the atoms (i.e. by replacing the variables by terms). E.g. the abstract state $hc(X)$, where X is a variable, could represent the set of states $\{hc(1), hc(2)\}$ depending on the terms (1 and 2 in this case) in the LOHMM. If the logical atom does not contain any variables such as $hc(1)$, it represents a singleton set. Abstract states are connected by *abstract transitions*, which summarize sets of transitions between states. When a transition is made, a state is sampled from the encompassing abstract state. Subsequently an observation symbol is generated in the same manner. We will explain these concepts on an example. For more details, we refer to [7].

3.1 An Example of a LOHMM

Fig. 1 shows an example of a LOHMM. The vertices in the model represent abstract (hidden) states where the predicate $hc(ID)$ (resp. $sc(ID)$) represents a block ID of consecutive helices (resp. strands). In such models, we find three different types of edges:

Solid edges between abstract states specify the abstract transitions. Transition probabilities and emission symbols are associated to them. An example transition from Fig. 1 is $sc(X) \overset{st(O,L):0.5}{\longleftarrow} hc(Y)$. Such a solid edge expresses that if one is in one of the states represented by $hc(Y)$ one will go to one of the states in $sc(X)$ with probability 0.5 while emitting a symbol in $st(O, L)$.

Dotted edges indicate that two abstract states behave in exactly the same way. If we follow a transition to an abstract state with an outgoing dotted edge, we will automatically follow that edge. Consider the dotted edge going from $sc(Z)$ to $sc(X)$ in Fig 1. The two abstract states are identical. The dotted edge is needed in this case because the variables appearing in the abstract states are different. We could not have written this using solid edges alone as the meaning of the solid edge $sc(X) \overset{st(O,L):0.5}{\longleftarrow} sc(X)$ is different from that of $sc(Z) \overset{st(O,L):0.5}{\longleftarrow} sc(X)$. Whereas the first transition only allows a transition between the same state say $sc(1)$ (because the X is identical), the second one allows transition between different states such as $sc(1)$ and $sc(2)$. In a logical sense, dotted edges implement a kind of recursion.

Dashed edges represent a kind of default reasoning. This is often used to model exceptions. Consider the dashed edge in Fig. 1 connecting $hc(Y)$ and $hc(2)$. This dashed edge denotes that $hc(2)$ is a more specific state than $hc(Y)$. This implies that the set of states represented by the more specific (abstract) state is a subset of that represented by the more general one. Logically speaking, the more specific state $hc(2)$ can be obtained by substituting Y by 2 in the more general state $hc(Y)$. Dashed edges and default reasoning are useful because they represent exceptions. Indeed, in our current example, the outgoing probability labels associated to $hc(2)$ are different from those for $hc(Y)$. This actually implies that the $hc(2)$ acts as an exception to the states represented by $hc(Y)$. So for $Y = 2$ we employ the transitions from $hc(2)$ and for $Y \neq 2$ we follow those indicated by $hc(Y)$.

Let us now explain how the model in Fig. 1 generates the sequence of observations $he(h(right, 3to10), 10)$, $st(plus, 10)$, $st(plus, 15)$, $he(h(right, alpha), 9)$, (cf. Fig. 2). Starting from the artificial state $start$, it chooses an initial abstract state, say $hc(1)$. Forced to follow the dotted edge, it enters the abstract state $hc(Y)$. In each abstract state, the model samples values for all variables that

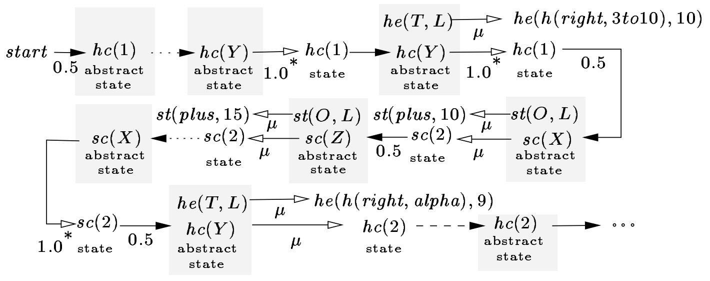


Figure 2: Generating the observation sequence $he(h(right, 3to10), 10)$, $st(plus, 10)$, $st(plus, 15)$, $he(h(right, alpha), 9)$ by the LOHMM in Fig. 1 (* $\mu = 1.0$ due to unification).

are not instantiated yet according to a *selection distribution* μ.

The function μ specifies for each abstract state a distribution over the possible instantiations of the abstract state. E.g. $\mu(he(h(right, alpha), 4) \mid he(h(T, alpha), 4)) = 0.5$ says that the model samples $he(h(right, alpha), 4)$ with probability 0.5 from $he(h(T, alpha), 4)$ whereas $\mu(he(h(right, alpha), 4) \mid he(h(T, A), 4)) = 0.05$ specifies that $he(h(right, alpha), 4)$ is sampled with probability 0.05 from $he(h(T, A), 4)$. In general, any probabilistic representation such as Markov chains or Bayesian networks might be used to represent μ. In our experiments, we followed a naïve Bayes approach, i.e. each argument of a predicate is assumed to be independent of the other arguments. E.g., to compute $\mu(he(h(right, alpha), 4) \mid he(T, L))$, we compute the product of $P_T(h(right, alpha))$ and $P_L(4)$.

Since the value of Y was already instantiated in the previous abstract state $hc(1)$, the model samples with probability 1.0 the state $hc(1)$. It selects the transition to $hc(Y)$ observing $he(T, L)$. Since Y is shared among the head and the body, the state *hc(1)* is selected with probability 1.0. The observation $he(h(right, 3to10), 10)$ is sampled from $he(T, L)$ using the selection distribution μ. Now, the model goes over to the abstract state $sc(X)$, emitting $st(plus, 10)$ which in turn was sampled from $st(O, L)$. Variable X in $sc(X)$ is not yet bound; so, a value, say 2, is sampled using μ. Next, we move on to abstract state $sc(Z)$, emitting $st(plus, 15)$. The variable Z is sampled to be 3. The dotted edge brings us back to $sc(X)$ and automatically unifies X with Z, which is bound to 3. Emitting $he(h(right, alpha), 9)$, the model returns to abstract state $hc(Y)$. Assume that it samples 2 for variable Y, it has to follow the dashed outgoing edge to $hc(2)$, which represents an exception to $hc(Y)$. This process is similar to unrolling dynamic Bayesian networks[2] and to grounding logic programs[8].

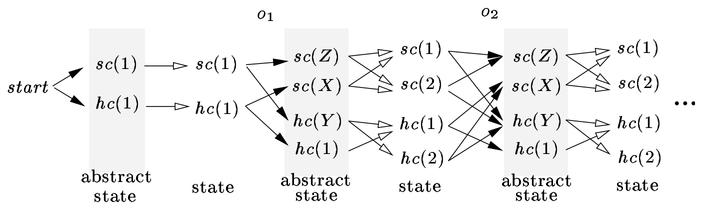


Figure 3: Illustration of the trellis induced by the LOHMM in Fig. 1. In contrast with HMMs, there is an additional layer where the states are sampled from abstract states.

3.2 Semantics and Evaluation

Each HMM is a LOHMM consisting of propositional, logical transitions only. Having the described grounding/unrolling process in mind, it is clear that a LOHMM defines a HMM given a *selection distribution* μ. There are a finite set of abstract transitions and each domain associated to an argument of a predicate is finite. Thus, the set of states and, therefore, the set of (ground) transitions is finite. To summarize, there are two primary differences to HMMs. First, transition probabilities are defined by a product of abstract transition probabilities and the selection probability. Second, the set of states represented by an abstract state can vary with the domains associated to predicates.

A trellis can be built as follows: After selecting an abstract transition, μ generates the relevant states from the head of the abstract transition (cf. Fig. 3). Based on the trellis, it is easy to adapt the *forward- backward*, the *Viterbi* and the *Baum-Welch* algorithms for HMMs to LOHMMs. E.g. in the forward-backward procedure, the probabilities α and β are computed for each reachable state (sets S_t) recursively. The $\alpha_t(s)$ is the probability of the partial observation sequence $o_1, \ldots, o_{t-1}$ and state s at time t given the LOHMM. The $\beta_t(s)$ is the probability of the partial observation sequence $o_t, \ldots, o_T$ given a state s at time t and the LOHMM. Set $\alpha_0(start) = 1.0$ and $\beta_T(s) = 1.0$ for every $s \in S_T$. Recursive formulae are $\alpha_t(h) = \sum_{cl} \sum_{b \in S_{t-1}} \alpha_{t-1}(b) p_{cl} p_\mu \delta(cl, b, h, o_t)$ and $\beta_t(b) = \sum_{cl} \sum_{h \in S_{t+1}} \beta_{t+1}(h) p_{cl} p_\mu \delta(cl, b, h, o_t)$, where *cl* is a transition in the LOHMM, p_{cl} is the transition probability and p_μ is the selection probability given by μ. The indicator function $\delta(cl, b, h, o_t) = 1$ whenever transition *cl* can take from state b to h observing o_t and the transition *cl* has the most specific body for b. The other algorithms can be adapted analogously.

4 Experiments

The aim of the experiments described below is to put the following hypotheses to test:

H1 LOHMMs are capable of distinguishing between different folds based on a logical representation of the secondary structure of domains.

H2 The inspection of LOHMMs reveal distinguishing features of folds.

H3 LOHMMs can be applied to real-world problems.

H4 In some applications in computational biology, LOHMMs are by at least an order of magnitude smaller than their instantiations which are HMMs.

We implemented the EM algorithm (with pseudocounts) using the Prolog system Sicstus-3.8.6. The experiments were ran on a Pentium-III-600 MHz machine. Our task was to classify sequences representing protein secondary structures into one of five folds. To do so, we followed the standard approach to classification based on HMMs. We chose a LOHMM (see Fig. 5), fixed its structure, and randomly generated for each fold a set of initial abstract transition probabilities and domain distributions. From each fold dataset [a] described in Section 2, we randomly sampled a training set consisting of 200 sequences. The remaining sequences were used as a test set. Then, we trained these five LOHMMs, one per fold. We used a simple, but common stopping criterion: EM stops if a change in log-likelihood is less than 10^{-1} from one iteration to the next. To evaluate the learned models, we computed the log-likelihood that each model gave to a sequence in the test sets. If the i-th model was the most likely one, then we classified the sequence as a member of class i.

The used LOHMM structure is given in Fig. 5. The hidden states are modelled using $hc(ID, T, L)$ and $sc(ID, O, L)$ representing blocks of consecutive helices and strands. Being in a block ID of consecutive helices (resp. strands), the model will remain in the block or transition to a new block $s(ID)$ of strands (resp. helices). This model takes into account type T, length L and orientation O information. Moreover, there are specific abstract transitions for helices of types $h(right, alpha)$ and $h(right, 3to10)$, and for parallel and anti-parallel strands, and for being at the beginning of a sheet. This enabled us to model the "process" within blocks of consecutive helices quite detailed, and of transitions from blocks of consecutive helices to strands and vice versa. The ID enables

[a]For the extraction of the Prolog facts from the PDB, we adapted the program `secondary.c` made available by the Learning and Planning group of the University of Texas at Arlington (`http://cygnus.uta.edu/subdue/databases/db/proteins.tar.gz`).

Table 1: Confusion matrix showing actual vs. predicted fold classification.

actual \ predicted	fold1	fold2	fold23	fold37	fold55
fold1	736	61	51	62	30
fold2	49	291	53	31	11
fold23	18	23	166	11	15
fold37	55	44	27	282	19
fold55	0	1	1	3	147

Table 2: Precision and recall for each fold rounded to second decimal.

	fold1	fold2	fold23	fold37	fold55
Precision	0.86	0.69	0.56	0.72	0.66
Recall	0.78	0.67	0.71	0.66	0.96

us to have general directed transitions from one block to exactly one successor block.

Results

Our implementation of EM took at most five iterations and approximately 5 minutes to estimate the maximum-likelihood parameters per fold. Given our quantization of the helix and strand lengths, the LOHMM consisted of 74 abstract transition and 46 domain distribution probabilities, whereas the corresponding HMM would consist of over $62,000$ transition probabilities. So, the abstract representation of states and transitions in LOHMMs achieves, by design, a remarkable compression of the model, which supports hypothesis H4.

The classification results are summarized by the confusion matrix in Table 1. In this section, the TIM beta/alpha-barrel fold will be denoted as $fold1$, the NAD(P)-binding Rossmann-fold as $fold2$, the Ribosomal protein L4 fold as $fold23$, the glucosamine 6-phosphate deaminase/isomerase fold as $fold37$, and the leucine aminopeptidas fold as $fold55$. In total, 74% (1622 out of 2187) sequences were correctly classified. This result is in the same range as the one reported by[11] (75%). However, we have to emphasize that the datasets are not completely comparable. In contrast to this result, a learner predicting always the majority class would achieve an predictive accuracy of 43%. These results suggest that hypothesis H1 holds. In Table 2, we also give our results in terms of the *recall* and *precision.* Recall is defined as the sum of true positives divided by the sum of true positives and false negatives. Precision is defined as the sum of true positives divided by the sum of true positives and false positives. As can be seen, the recall and precision figures vary among the folds, but within the folds recall and precision are well balanced. In other words, a good

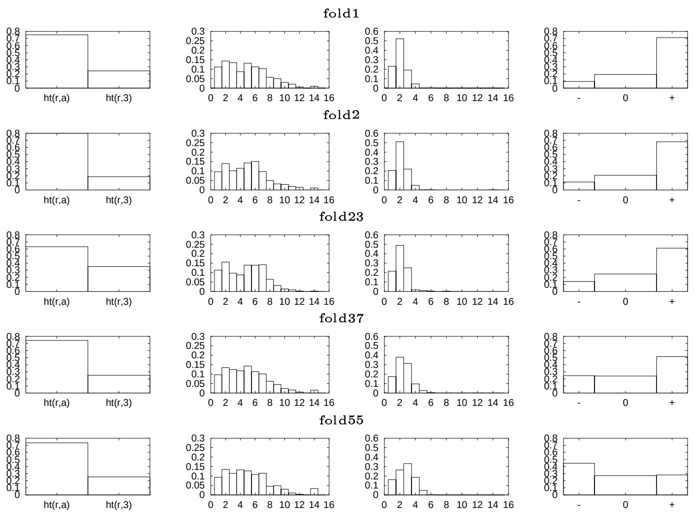


Figure 4: Estimated selection distributions for the five folds (from left to right: helix types, helix lengths, strand lengths, and strand orientations). The distributions specify the probability that a state is sampled from an abstract state (if needed) using a naïve Bayes scheme. The distribution over helix types shows, that only the types *ht*(*right*, *alpha*) (shortly *ht*(*r*, *a*)) and *ht*(*right*, *3to10*) (shortly *ht*(*r*, 3)) occurred in the data. Due to pseudocounts, no probability value is zero.

precision is not bought at the expense of a good recall, and vice versa. The relatively low precision values for *fold23* and *fold55* are explained by a smaller number of test examples for these two folds. Finally, we inspected the trained LOHMM for characteristic differences. More precisely, we plotted for each of the five estimated LOHMMs the probability distributions implicitly defining μ (see Fig. 4) following a naïve Bayes scheme. Please note that μ defines the probability of sampling a state from an abstract state taking variable binding into account, i.e. that μ and therefore the distributions in Fig. 4 depend on the logical structure of the LOHMM. Upon visual inspection, differences can be found as follows (hypothesis H2):

- Regarding the helix types, $fold23$ differs from the others in that the probability of selecting right-handed alpha helices seems to be lower. Also, the probability of right-handed 3to10 helices to be selected seems to be higher than for the other folds.
- According to the strand lengths, we can group the first three and the last

two folds.

- As for strand orientations, we have uniform "patterns" for $fold1$, $fold2$ and $fold23$, but characteristic patterns for $fold37$ and $fold55$.

To summarize, we believe that the results obtained in our experiments are quite promising also for what concerns the application domain. Therefore, they indicate that the answer to hypothesis H3 should be positive.

5 Related Work

Gough et al. [6] presented an approach to sequence annotation based on profile HMMs trained on the primary structure of domains for each superfamily in SCOP. The present study is at a different level of abstraction: Firstly, we are working with a secondary structure representation, and not a primary structure representation. Secondly, we are dealing with SCOP folds, not SCOP superfamilies. It might be interesting to apply our approach also at the (more detailed) superfamily level. The goal in [6] was to annotate sequences based on a library of HMMs that represent all proteins of known structure. In contrast, our short-term goal was to give a proof of the principle, with the intermediate-term goal of providing a tool that helps to gain insights into structural characteristics.

Turcotte et al. [12,11] applied the Machine Learning and Inductive Logic Programming (ILP) tool Progol to a similar task as the one tackled in this paper. The task there was also to predict SCOP folds based on a high-level logical representation. The difference is that we are working with a larger, more recent dataset, a different representation, and that we are applying a different Machine Learning approach based on probability theory.

HMMs have been extended in a number of different ways e.g. hierachichal HMMs [3], factorial HMMs [5] and based on tree automata [4]. Non of them utilize logical representations. *Relational Markov Models* (RMMs), which were recently introduced and applied to web navigation by Anderson *et. al* [1], are an exception. RMMs do not allow for variable binding, unification nor hidden states.

6 Conclusion

In this paper, we have introduced Logical Hidden Markov Models (LOHMMs) and applied them to the task of finding structural signatures of protein folds. LOHMMs offer the possibility to specify states and transitions at an abstract level, and thereby offer a significant reduction in model size compared to regular HMMs. Our experiments show that the learning performance of LOHMMs is good. We have also shown that it is easy to extract characteristic patterns from the learned models.

In the future, we will conduct more experiments on more folds in SCOP. Furthermore, we are currenlty developing algorithms for learning the (logical) structure of LOHMMs.

Acknowledgements This research was partly supported by the European Union IST programme under contract number IST-2001-33053, *APrIL*. T. Raiko was supported by a Marie Curie fellowship at DAISY, HPMT-CT-2001-00251.

1. C. R. Anderson, P. Domingos, and D. S. Weld. Relational Markov Models and their Application to Adaptive Web Navigation. In *Proceedings of KDD-2002*, July 2002.
2. T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *Proceedings of AAAI-88*, 1988.
3. S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: analysis and applications. *Machine Learning*, 32, 1998.
4. P. Frasconi, G. Soda, and A. Vullo. Hidden markov models for text categorization in multi-page documents. *Journal of Intelligent Information Systems*, 18(2/3):195–217, 2002.
5. Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.
6. J. Gough, K. Karplus, R. Hughey, and C. Chothia. Assignment of homology to genome sequences using a library of hidden markov models that represent all proteins of known structure. *Journal of Molecular Biology*, 313(4):903–919, 2001.
7. K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden markov models. Tech. Rep. 175, University of Freiburg, June 2002.
8. J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1989.
9. L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2), 1989.
10. L. R. Rabiner and B. H. Juang. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, January:4–16, 1986.
11. M. Turcotte, S. Muggleton, and M. J. E. Sternberg. Discovery of structural signatures of protein fold and function. *Journal of Molecular Biology*, 306(3):591–605, 2001.
12. M. Turcotte, S. Muggleton, and M. J. E. Sternberg. The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Machine Learning*, 43(1/2):81–95, 2001.

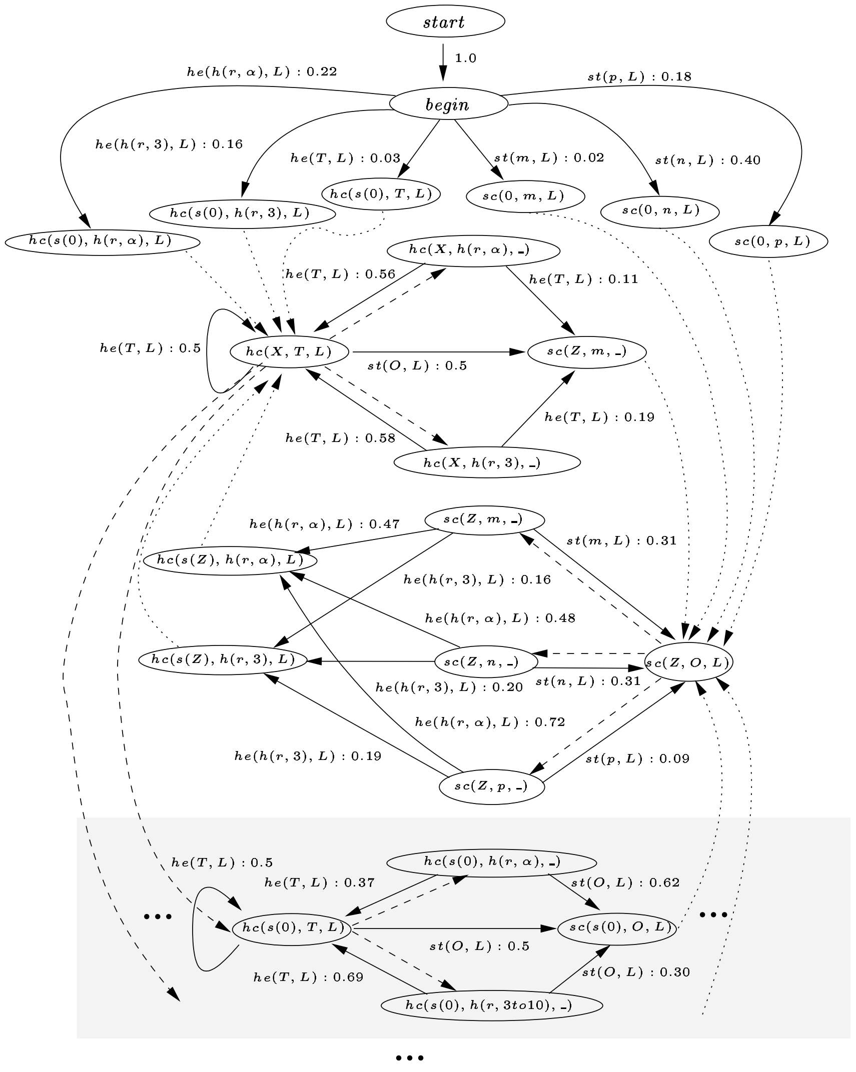


Figure 5: The estimated logical (hidden) Markov model of fold 1. The *end* state is omitted. If probabilities do not sum to 1.0, then there is a transition to *end*. The symbol _ denotes anonymous variables which are read and treated as distinct, new variables each time they are encountered. There are copies of the shaded part for $hc(s^2(0), T, L), \ldots, hc(s^7(0), T, L)$. Terms are abbreviated using their starting alphanumerical and α for *alpha*.

Publication 9

K. Kersting and T. Raiko. 'Say EM' for Selecting Probabilistic Models for Logical Sequences. In the *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI 2005)*, pp. 300–307, Edinburgh, Scotland, July 26–29, 2005.

’Say EM’ for Selecting Probabilistic Models for Logical Sequences

Kristian Kersting
Machine Learning Laboratory
University of Freiburg
Georges-Koehler-Allee 079, 79112 Freiburg, Germany

Tapani Raiko
Laboratory of Computer and Information Science
Helsinki University of Technology
P.O. Box 5400, 02015 HUT, Finland

Abstract

Many real world sequences such as protein secondary structures or shell logs exhibit a rich internal structures. Traditional probabilistic models of sequences, however, consider sequences of flat symbols only. *Logical hidden Markov models* have been proposed as one solution. They deal with logical sequences, i.e., sequences over an alphabet of logical atoms. This comes at the expense of a more complex model selection problem. Indeed, different abstraction levels have to be explored. In this paper, we propose a novel method for selecting logical hidden Markov models from data called SAGEM. SAGEM combines *generalized expectation maximization*, which optimizes parameters, with structure search for model selection using *inductive logic programming* refinement operators. We provide convergence and experimental results that show SAGEM’s effectiveness.

1 Introduction

Hidden Markov models [21] (HMMs) are extremely popular for analyzing sequential data. Areas of application include computational biology, user modeling, and robotics. Despite their successes, HMMs have a major weakness: they handle only sequences of flat, i.e., unstructured symbols. In many applications the symbols occurring in sequences are structured. Consider, e.g., the sequence of UNIX commands `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, ... Such data have been used to train HMMs for anomaly detection [15]. However, as the above command sequence shows, UNIX commands may have parameters. Thus, commands are essentially *structured* symbols. HMMs cannot easily deal with this type of structured sequences. Typically, the application of HMMs requires either 1) ignoring the structure of the commands (i.e., the parameters), or 2) taking all possible parameters explicitly into account. The former approach results in serious information loss; the latter in a combinatorial explosion in the number of parameters and, as a consequence, inhibits generalization.

The above sketched problem with HMMs is akin to the problem of dealing with structured examples in traditional machine learning algorithms as studied in the field of inductive logic programming (ILP) [17]. Recently, Kersting et al. [12] proposed *logical HMMs* (LOHMMs) as an *probabilistic* ILP [4] framework that upgrades HMMs to deal with structure. The key idea is to employ logical atoms. Using logical atoms, the above UNIX command sequence can be represented as $\mathtt{emacs}(\mathtt{lohmms.tex}), \mathtt{ls}, \mathtt{latex}(\mathtt{lohmms.tex}), \ldots$ LOHMMs have been proven to be useful within bioinformatics domains. For instance in [12], the LOHMMs used to discover structural signatures of protein folds were simpler but more effective compared to corresponding HMMs (120 vs. > 62000 parameters). The compactness and comprehensibility, however, comes at the expense of a more complex model selection problem. So far, model selection for LOHMMs has not been investigated. Our main contribution is SAGEM, German for ’say EM’, a novel method for selecting LOHMM structures from data. Selecting a structure is a significant problem for many reasons. First, eliciting LOHMMs from experts can be a laborious and expensive process. Second, HMMs are commonly learned by estimating the maximum likelihood parameters of a fixed, fully connected model. Such an approach is not feasible for LOHMMs as different abstraction levels have to be explored. Third, LOHMMs are strictly more expressive than HMMs. In [11], LOHMMs are used to classify tree-structured mRNA data. Finally, the parameter estimation of a LOHMM is a costly nonlinear optimization problem, so the naïve search is infeasible.

SAGEM adapts Friedman’s *structural EM* [6]. It combines a *generalized expectation maximization* (GEM) algorithm, which optimizes parameters, with structure search for model selection using ILP refinement operators. Thus, SAGEM explores different abstraction levels due to ILP refinement operators, and, due to a GEM approach, it reduces the selection problem to a more efficiently solvable one.

The outline of the paper is as follows. Section 2 reviews LOHMMs and their underlying logical concepts; Section 3 formalizes the model selection problem; in Section 4, we present a naïve learning algorithm; in Section 5, we introduce a structural, generalized EM – called SAGEM – for learning LOHMMs. SAGEM is experimentally evaluated in Section 6. Before concluding we discuss related work.

2 Probabilistic Models for Logical Sequences

We will briefly review *logical Hidden Markov models* (LOHMMs) [12, 13, 11]. The logical component of HMMs corresponds to a *Mealy machine*, i.e., to a finite state machine where the output symbols are associated with transitions. The key idea to develop probabilistic models for structured sequences is to replace these flat symbols by abstract symbols, more precisely logical atoms.

First-Order Predicate Logic: A *first-order logic alphabet* Σ is a set of relation symbols $\mathtt{r}$ with arity $m \geq 0$, written $\mathtt{r/m}$, and a set of function symbols $\mathtt{f}$ with arity $n \geq 0$, written $\mathtt{f/n}$. An *atom* $\mathtt{r(t_1,\ldots,t_m)}$ is a relation symbol $\mathtt{r}$ followed by a bracketed m-tuple of terms $\mathtt{t_i}$. A *term* is a variable $\mathtt{V}$ or a function symbol $\mathtt{f}$ of arity n immediately followed by a bracketed n-tuple of terms $\mathtt{s_j}$, i.e., $\mathtt{f(s_1,\ldots,s_n)}$. A *definite clause* $\mathtt{A} \leftarrow \mathtt{B}$ consists of atoms A and B and can be read as $\mathtt{A}$ *is true if* $\mathtt{B}$ *is true*. A substitution $\theta = \{\mathtt{V_1/t_1},\ldots,\mathtt{V_k/t_k}\}$, e.g. $\{\mathtt{X/tex}\}$, is an assignment of terms $\mathtt{t_i}$ to variables $\mathtt{V_i}$. Applying a substitution θ to a term, atom or clause e yields the instantiated term, atom, or clause $e\theta$ where all occurrences of the variables $\mathtt{V_i}$ are simultaneously replaced by the term $\mathtt{t_i}$, e.g. $\mathtt{ls(X)} \leftarrow \mathtt{emacs(F,X)}\{\mathtt{X/tex}\}$ yields $\mathtt{ls(tex)} \leftarrow \mathtt{emacs(F,tex)}$. A term, atom or clause e is called *ground* when it contains no variables, i.e., $vars(e) = \emptyset$. The *Herbrand base* of Σ, denoted as hb_Σ, is the set of all ground atoms constructed with the predicate and function symbols in Σ. The set $G_\Sigma(\mathtt{A})$ of an atom $\mathtt{A}$ consists of all ground atoms $\mathtt{A}\theta$ belonging to hb_Σ.

Our running example will be user modeling. For example, `emacs(readme, other)` means that the user of type `other` writes a command `emacs readme` to a shell.

Logical Hidden Markov Models (LOHMMs): The sequences generated by LOHMMs are sequences of ground atoms rather than flat symbols. Within LOHMMs, the flat symbols employed in traditional HMMs are replaced by logical atoms such as $\mathtt{emacs(F,tex)}$. Each atom $\mathtt{emacs(F,tex)}$ there represents the set of ground atoms $G_\Sigma(\mathtt{emacs(F,tex)})$, e.g. $\mathtt{emacs(readme,tex)} \in G_\Sigma(\mathtt{emacs(F,tex)})$.

Additionally, we assume that the alphabet is typed which in our case means that there is a function mapping every predicate $\mathtt{r/m}$ and number $1 \leq i \leq m$ to the set of ground terms allowed as the i-th argument of predicate $\mathtt{r/m}$. This set is called the domain of the i-th argument of predicate $\mathtt{r/m}$.

Figure 1 shows a LOHMM graphically. The states, observations, and transitions of LOHMMs are **abstract** in the sense that every abstract state or observation $\mathtt{A}$ represents all possible concrete states in $G_\Sigma(\mathtt{A})$. In Figure 1 *solid edges* encode **abstract transitions**. Let $\mathtt{H}$ and $\mathtt{B}$ be logical atoms representing abstract states, let $\mathtt{O}$ be a logical atom representing an abstract output symbol. An abstract transition from state $\mathtt{B}$ with probability p to state $\mathtt{H}$ and omitting $\mathtt{O}$ is denoted by $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$. If $\mathtt{H}$, $\mathtt{B}$, and $\mathtt{O}$ are all ground, there is no difference to 'normal' transitions. Otherwise, if $\mathtt{H}$, $\mathtt{B}$, and $\mathtt{O}$ have no variables in common, the only difference to 'normal' transitions is that for each abstract state (resp. observation) we have to sample which concrete state (resp. observation) we are in. Otherwise, we have to remember the variable bindings. More formally, let $\mathtt{B}\theta_\mathtt{B} \in G_\Sigma(\mathtt{B})$, $\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H} \in G_\Sigma(\mathtt{H}\theta_\mathtt{B})$ $\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}\theta_\mathtt{O} \in G_\Sigma(\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H})$, and let μ be a **selection distribution**. Then with probability $p \cdot \mu(\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H} \mid \mathtt{H}\theta_\mathtt{B}) \cdot \mu(\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}\theta_\mathtt{O} \mid \mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H})$, the model makes a transition from state $\mathtt{B}\theta_\mathtt{B}$ to $\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H}$ and emits symbol $\mathtt{O}\theta_\mathtt{B}\theta_\mathtt{H}\theta_\mathtt{O}$.

A selection distribution specifies for each abstract state (respectively observation) $\mathtt{A}$ over the alphabet Σ a distribution $\mu(\cdot \mid \mathtt{A})$ over $G_\Sigma(\mathtt{A})$. Consider, for example, the abstract transition $0.5 : \mathtt{s(f(Z))} \xleftarrow{\mathtt{o(X,Y,Z)}} \mathtt{s(X)}$. Suppose, $\mathtt{B}\theta_\mathtt{B} = \mathtt{s(1)}$, $\mu(\mathtt{s(f(3))} \mid \mathtt{s(f(Z))}) = 0.2$ and $\mu(\mathtt{o(1,2,3)} \mid \mathtt{o(1,Y,3)}) = 0.05$. Then, from state `s(1)` with probability $0.5 \times 0.2 \times 0.05 = 0.005$ the output symbol is `o(1,2,3)` and the next state is `s(f(3))`. To reduce the model complexity, we employ a naïve Bayes approach in which – at the expense of a lower expressivity – functors are neglected and variables are treated independently. More precisely, for each domain D_i there is a probability distribution P_{D_i}. Let $\mathrm{vars}(\mathtt{A}) = \{\mathtt{V_1},\ldots,\mathtt{V_l}\}$ be the variables occurring in $\mathtt{A}$, and let $\theta = \{\mathtt{t_1/V_1},\ldots\mathtt{t_l/V_l}\}$ be a substitution grounding $\mathtt{A}$. Each $\mathtt{V_j}$ is then considered a random variable over the domain of the first argument of $\mathtt{r/m}$ it appears in, denoted by $D_{\mathtt{V_j}}$. Then, $\mu(\mathtt{A}\theta \mid \mathtt{A}) = \prod_{j=1}^{l} P_{D_{\mathtt{V_j}}}(\mathtt{V_j} = \mathtt{t_j})$. For instance, $\mu(\mathtt{s(f(3))} \mid \mathtt{s(f(Z))})$ equals $P_1^{\mathtt{s}/1}(3)$.

Indeed, multiple abstract transitions can match a given ground state. Consider the abstract states $\mathtt{B_1} = \mathtt{emacs(File,tex)}$ and $\mathtt{B_2} = \mathtt{emacs(File,User)}$ in Fig. 1 **(a)**. The abstract state $\mathtt{B_1}$ is more specific than $\mathtt{B_2}$ because there exists a substitution $\theta = \{\mathtt{User/tex}\}$ such that $\mathtt{B_2}\theta = \mathtt{B_1}$, i.e., $\mathtt{B_2}$ subsumes $\mathtt{B_1}$. Therefore $G_\Sigma(\mathtt{B_1}) \subseteq G_\Sigma(\mathtt{B_2})$ and the first transition can be regarded as more informative than the second one. It should therefore be preferred over the second one when starting

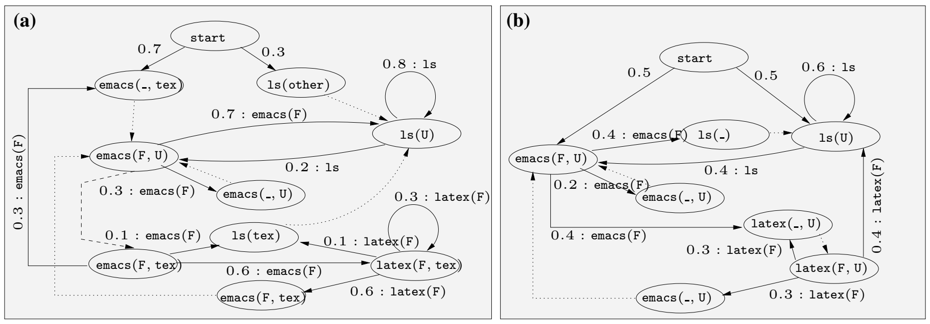


Figure 1: Logical hidden Markov models. The vertices represent abstract (hidden) states. Solid edges encode abstract transitions. Dotted edges indicate that two abstract states behave in exactly the same way. Dashed edge denote the *more general than* relation. The LOHMMs are described in the text.

e.g. from $\mathtt{emacs(hmm1, tex)}$. We will also say that the *transitions* of the first abstract state are *more specific* than the second ones; encoded by *dashed edges*. These considerations lead to the **conflict resolution strategy** [1] of only considering the maximally specific transitions that apply to a state in order to determine the successor states. This implements a kind of exception handling or default reasoning and is akin to Katz's *back-off* n-gram models [10]. In back-off n-gram models, the most detailed model that is deemed to provide sufficiently reliable information about the current context is used. That is, if one encounters an n-gram that is not sufficiently reliable, then back-off to use an $(n-1)$-gram; if that is not reliable either then back-off to level $n-2$, etc.

Finally, *dotted edges* denote that two abstract states behave in exactly the same way. If we follow a transition to an abstract state with an outgoing dotted edge, we will automatically follow that edge making appropriate unifications.

Definition 1 *A* logical hidden Markov model *(LOHMM) is a tuple $M = (\Sigma, \mu, \Delta)$ where Σ is a logical alphabet, μ a selection probability over Σ and Δ is a set of abstract transitions. Let $\mathbf{B}$ be the set of all atoms that occur as the body part of transitions in Δ. We require*

$$\forall \mathtt{B} \in \mathbf{B} : \sum\nolimits_{p:\mathtt{H} \xleftarrow{\mathtt{o}} \mathtt{B} \in \Delta} p = 1. \qquad (1)$$

In [11] it is proven that LOHMMs specify a unique probability measure over hb_Σ. Here, we would like to exemplify that LOHMMs are generative models. Consider the model in Fig. 1**(a)**. Starting from $\mathtt{start}$, it chooses an initial abstract state, say $\mathtt{emacs(\_, tex)}$ with probability 0.7. Here, $\_$ denotes an anonymous variable which is read and treated as distinct, new variables each time it is encountered. Forced to follow the dotted edge, it enters the abstract state $\mathtt{emacs(F, U)}$. In each abstract state, the model samples values for all variables that are not instantiated yet according to the *selection distribution* μ. Since the value of $\mathtt{U}$ was already instantiated in the previous abstract state $\mathtt{emacs(F, tex)}$, the model has only to sample a value for $\mathtt{F}$, say $\mathtt{f1}$, using μ. Now, it selects a transition, say, to $\mathtt{latex(F, tex)}$ with probability 0.6. Since $\mathtt{F}$ is shared among the head and the body, the state $\mathtt{latex(f1, tex)}$ is selected with probability 1.0. The observation $\mathtt{emacs(f1)}$ is emitted from $\mathtt{emacs(F)}$ with probability 1.0 using μ. Now, the model goes over to, say $\mathtt{ls(tex)}$, emitting $\mathtt{latex(f1)}$ which in turn was sampled from $\mathtt{latex(F)}$. The dotted edge brings us to $\mathtt{ls(U)}$ and automatically unifies $\mathtt{U}$ with $\mathtt{tex}$. Emitting $\mathtt{ls}$, we return to $\mathtt{emacs(F, tex)}$ where $\mathtt{F}$ now denotes a new filename.

3 The Learning Setting

For traditional HMMs, the learning problem basically collapses to parameter estimation (i.e., estimating the transition probabilities) because HMMs can be considered to be fully connected. For LOHMMs, however, we have to account for different abstraction levels. The model selection problem can formally be defined as:

Given a set $\mathbf{O} = \{O_1, \ldots, O_m\}$ of data cases independently sampled from the same distribution, a set $\mathcal{M}$ of LOHMMs, and a scoring function *score*$_\mathbf{O} : \mathcal{M} \mapsto \mathbb{R}$, **find** a hypothesis $M^* \in \mathcal{M}$ that maximizes *score*$_\mathbf{O}$.

Each *data case* $O_i \in \mathbf{O}$ is a sequence $O_i = \mathtt{o}_{\mathtt{i},1}\mathtt{o}_{\mathtt{i},2} \ldots \mathtt{o}_{\mathtt{i},\mathtt{T}}$ of ground atoms and describes the observations evolving over time. For instance in the user modeling domain a data case could be $\mathtt{emacs(lohmms), ls, emacs(lohmms)}$. The corresponding evolution of the system's state over time $H_i = \mathtt{h}_{\mathtt{i},0}\mathtt{h}_{\mathtt{i},1} \ldots \mathtt{h}_{\mathtt{i},\mathtt{T_i}+1}$ is hidden, i.e. , not specified in O_i. For instance, we do not know whether $\mathtt{emacs(lohmms)}$ has been generated by $\mathtt{emacs(lohmms, prog)}$ or $\mathtt{emacs(lohmms, tex)}$.

The *hypothesis space* $\mathcal{M}$ consists of all candidate LOHMMs to be considered during search. We assume Σ

[1]Another conflict resolution strategy would be *smoothing*, i.e. , considering all matching abstract states. We chose not to use smoothing to keep the LOHMM locally interpretable, i.e. to have a single abstract body for each ground state.

to be given. Thus, the possible constants which can be selected by μ are apriori known. Each $M \in \mathcal{M}$ is parameterized by a vector $\boldsymbol{\lambda}_M$. Each (legal) choice of $\boldsymbol{\lambda}_M$ defines a probability distribution $P(\cdot \mid M, \boldsymbol{\lambda}_M)$ over $\mathrm{hb}(\Sigma)$. For the sake of simplicity, we will denote the underlying logic program (i.e., the set of abstract transitions without associated probability values) by M and abbreviate $\boldsymbol{\lambda}_M$ by $\boldsymbol{\lambda}$ as long as no ambiguities will arise. Furthermore, a syntactic bias on the transitions to be induced is a parameter of our framework, as usual in ILP [18]. For instance in the experiments, we only consider transitions which obey the type constraints induced by the predicates.

As score, we employ $\mathit{score}_{\mathbf{O}}(M, \boldsymbol{\lambda}) = \log P(\mathbf{O} \mid M, \boldsymbol{\lambda}) - \mathit{Pen}(M, \boldsymbol{\lambda}, \mathbf{O})$. Here, $\log P(\mathbf{O} \mid M, \boldsymbol{\lambda})$ is the *log-likelihood* of the current of model $(M, \boldsymbol{\lambda})$. It holds that the higher the log-likelihood, the closer $(M, \boldsymbol{\lambda})$ models the probability distribution induced by the data. The second term, $\mathit{Pen}(M, \boldsymbol{\lambda}, \mathbf{O})$, is a penalty function that biases the scoring function to prefer simpler models. Motivated by the *minimum description length* score for Bayesian networks, we use the simple penalty $\mathit{Pen}(M, \boldsymbol{\lambda}, \mathbf{O}) = |\Delta| \log(m)/2$. It is independent of the model parameters and therefore it can be neglected when estimating parameters. We assume that each M covers all possible observation sequences (over the given language Σ). This guarantees that all new data cases will get a positive likelihood.

4 A Naïve Learning Algorithm

A simple way of selecting a model structure is the following greedy approach:

```
1:  Let $\boldsymbol{\lambda}^0 = \mathrm{argmax}_{\boldsymbol{\lambda}}\ \mathit{score}_{\mathbf{O}}(M^0, \boldsymbol{\lambda})$
2:  **Loop** for $k = 0, 1, 2, \ldots$
3:      **Find** model $M^{k+1} \in \rho(M^k)$ that maximizes
            $\max_{\boldsymbol{\lambda}}\ \mathit{score}_{\mathbf{O}}(M^{k+1}, \boldsymbol{\lambda})$
4:      Let $\boldsymbol{\lambda}^{k+1} = \mathrm{argmax}_{\boldsymbol{\lambda}}\ \mathit{score}_{\mathbf{O}}(M^{k+1}, \boldsymbol{\lambda})$
5:  **Until** convergence, i.e., no improvement in score
```

It takes as input an initial model M^0 and the data $\mathbf{O}$. At each stage k we choose a model structure and parameters among the current best model M^k and its neighbors $\rho(M^k)$ (see below) that have the highest score. It stops, when there is no improvement in score. In practice, we initialize the parameters of each model on lines 1 and 3 randomly.

We will now show how to traverse the hypotheses space and how to estimate parameters for a hypothesis in order to score it. That is, we will make line 3 more concrete.

Traversing the Hypotheses Space: An obvious candidate for the initial hypothesis M^0 (which we also used in our experiments) is the fully connected LOHMM built over all maximally general atoms over Σ, i.e., expressions of the form $\mathtt{r}(\mathtt{X_1}, ..., \mathtt{X_m})$, where the $\mathtt{X_i}$ are different variables.

Now, to traverse the hypothesis space $\mathcal{M}$, we have to compute all neighbors of the currently best hypothesis M^k. To do so, we employ refinement operators traditionally used in ILP. More precisely, for the language bias considered and the experiments conducted in the present paper, we used the refinement operator $\rho : \mathcal{M} \mapsto 2^{\mathcal{M}}$ which selects a single clause $\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \mathcal{M}$ and adds a minimal specialization $\mathrm{cl}' \equiv p : \mathtt{H}' \xleftarrow{\mathtt{O}'} \mathtt{B}'$ of cl to $\mathcal{M}$ (w.r.t. to θ-subsumption). Specializing a single abstract transition means instantiating or unifying variables, i.e., $\mathrm{cl}' \equiv \mathrm{cl}\,\theta$ for some substitution θ. When adding cl' to M^k, we have to ensure that (1) the same observation and hidden state sequences are still covered and (2) the list of bodies $\mathbf{B}'$ after applying $\rho(M)$ should remain well-founded, that is, for each ground state, there is a unique maximally specific body in $\mathbf{B}'$. Both conditions together guarantee that the most specific body corresponding to a state always exists and is unique. Condition (1) can only be violated if $\mathtt{B}' \notin \mathbf{B}$. In this case, we add transitions with $\mathtt{B}'$ and maximally general heads and observations. Condition (2) is established analogously. We complete the keep the list of bodies well-founded by adding new bodies (and therefore abstract transitions) in a similar way as described above.

Consider refining the LOHMM in Fig. 1 **(b)**. When adding $\mathtt{ls(U)} \xleftarrow{\mathtt{latex(lohmm)}} \mathtt{latex(lohmm, U)}$, hence introducing the more specific abstract state $\mathtt{latex(lohmm, U)}$, further variants of the same abstract transition but with different heads have to be added. Otherwise condition (1) would be violated as the resulting LOHMM does not cover the same sequences as the original one; the state $\mathtt{latex(lohmm, U)}$ can only be left via $\mathtt{ls(U)}$ and not e.g. via $\mathtt{emacs(\_, U)}$. On the other hand, we have to be careful when subsequently adding abstract transitions for the body $\mathtt{latex(F, tex)}$. The problem is that we do not know which abstract body to select in state $\mathtt{latex(lohmm, tex)}$. To fulfill condition (2), you need to add abstract transitions for an additional, third abstract state $\mathtt{latex(lohmm, tex)}$, too.

Parameter Estimation: In the presence of hidden variables maximum log-likelihood (ML) parameter estimation is a numerical optimization problem, and all known algorithms involve nonlinear, iterative optimization and multiple calls to an inference algorithm. The most common approach for HMMs is the Baum-Welch algorithm, an instance of the EM algorithm [5]. In each iteration $l + 1$ it performs two steps:

(E-step) *Compute the expectation of the log-likelihood given the old model* $(M^k, \boldsymbol{\lambda}^{k,l})$ *and the observed data* $\mathbf{O}$*, i.e.,* $Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) = E\left[\log P(\mathbf{O}, \mathbf{H} \mid M^k, \boldsymbol{\lambda}) \mid M^k, \boldsymbol{\lambda}^{k,l}\right]$.

Here, $\mathbf{O}, \mathbf{H}$ denotes the completion of $\mathbf{O}$ where the evolution $\mathbf{H}$ of the system's state over time is made explicit. The current model $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$ give us the conditional distribution governing $\mathbf{H}$, and $E[\cdot|\cdot]$ denotes the expectation over it. The function Q is called the *expected score*.

(M-step) *Maximize the expected score* $Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *w.r.t.* $\boldsymbol{\lambda}$, *i.e.,* $\boldsymbol{\lambda}^{k,l+1} = \mathit{argmax}_{\boldsymbol{\lambda}}\, Q(M^k, \boldsymbol{\lambda}|M^k, \boldsymbol{\lambda}^{k,l})$.

The naïve greedy algorithm can easily be instantiated using the EM. The problem, however, is its huge computational costs. To evaluate a single neighbor, the EM has to run for a reasonable number of iterations in order to get reliable ML estimates of $\boldsymbol{\lambda}^{k'}$. Each EM iteration requires a full LOHMM inference on all data cases. In total, the running time per neighbor evaluation is at least $\mathcal{O}(\#\mathit{EM\ iterations} \cdot \mathit{size\ of\ data})$.

5 SAGEM: Structural Generalized EM

To reduce the computational costs, SAGEM (German for *'say EM'*) adapts Friedman's *structural EM* (SEM) [6]. That is, we take our current model $(M^k, \boldsymbol{\lambda}^k)$ and run the EM algorithm for a while to get reasonably completed data. We then fix the completed data cases and use them to compute the ML parameters $\boldsymbol{\lambda}^{k'}$ of each neighbor $M^{k'}$. We choose the neighbor with the best improvement of the score as $(M^{k+1}, \boldsymbol{\lambda}^{k+1})$ and iterate. More formally, we have

```
1:  Initialize $\boldsymbol{\lambda}^{0,0}$ randomly
2:  Loop for $k = 0, 1, 2, \ldots$
3:    Loop for $l = 0, 1, 2, \ldots$
4:      Let $\boldsymbol{\lambda}^{k,l+1} = \text{argmax}_{\boldsymbol{\lambda}}\, Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$
5:    Until convergence or $l = l_{\max}$
6:    Find model $M^{k+1} \in \rho(M^k)$ that maximizes
          $\max_{\boldsymbol{\lambda}} Q(M^{k+1}, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$
7:    Let $\boldsymbol{\lambda}^{k+1,0} = \text{argmax}_{\boldsymbol{\lambda}}\, Q(M^{k+1}, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$
8:  Until convergence
```

The hypotheses space is traversed as described in Section 4, and again we stop if there is no improvement in score. The following theorem shows that even when the structure changes in between, improving the expected score Q always improves the log-likelihood as well.

Theorem 1 *If* $Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) > Q(M^k, \boldsymbol{\lambda}^{k,l} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds, then* $\log P(\mathbf{O} \mid M, \boldsymbol{\lambda}) > \log P(\mathbf{O} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds.*

The proof is a simple extension of the argumentation by [16, Section 3.2 ff.]. To apply the algorithm to selecting LOHMMs, we will now show how to choose the best neighbour [2] in line 6.

[2] In the following, we will omit some derivation steps due to space restriction. They can be found in [13]. Furthermore, for the sake of simplicity, we will not explicitly check that a transition is *maximally specific* for ground states.

Let $c(\mathtt{b}, \mathtt{h}, \mathtt{o})$ denote the number of times the systems proceeds from ground state $\mathtt{b}$ to ground state $\mathtt{h}$ emitting ground observation $\mathtt{o}$. The expected score in line 6 simplifies to

$$Q(M, \boldsymbol{\lambda}|M^k, \boldsymbol{\lambda}^{k,l}) \qquad (2)$$
$$= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} \underbrace{E\left[c(\mathtt{b},\mathtt{h},\mathtt{o}) \middle| M^k, \boldsymbol{\lambda}^{k,l}\right]}_{=:ec(\mathtt{b},\mathtt{h},\mathtt{o})} \cdot \log P(\mathtt{h},\mathtt{o}|\mathtt{b}, M, \boldsymbol{\lambda}) \ .$$

The term $ec(\mathtt{b}, \mathtt{h}, \mathtt{o})$ in (2) denotes the expected counts of making a transition from ground state $\mathtt{b}$ to ground state $\mathtt{h}$ emitting ground observation $\mathtt{o}$. The expectation is taken according to $(M^k, \boldsymbol{\lambda}^{k,l})$.

An analytical solution, however, of the M-step in line 7 seems to be difficult. In HMMs, the updated transition probabilities are simply directly proportional to the expected number of times they are used. In LOHMMs, however, there is an ambiguity: multiple abstract transitions (with the same body), can match the same ground transition $(\mathtt{b}, \mathtt{h}, \mathtt{o})$. Using *ec* as sufficient statistics makes the M step nontrivial. The solution is to improve (2) instead of maximizing it. Such an approach is called *generalized EM* [16]. To do so, we follow a gradient-based optimization technique. We iteratively compute the *gradient* $\nabla_{\boldsymbol{\lambda}}$ of (2) w.r.t. the parameters of a LOHMM, and, then, take a step in the direction of the gradient to the point $\boldsymbol{\lambda} + \delta\nabla_{\boldsymbol{\lambda}}$ where δ is the step-size.

For LOHMMs, the gradient w.r.t. (2) consists of partial derivatives w.r.t. abstract transition probabilities and to selection probabilities. Assume that λ is the transition probability associated with some abstract transition cl. Now, the partial derivative of (2) w.r.t. some parameter λ is

$$\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial \lambda}$$
$$= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} ec(\mathtt{b},\mathtt{h},\mathtt{o}) \cdot \frac{\partial \log P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})}{\partial \lambda}$$
$$= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} \frac{ec(\mathtt{b},\mathtt{h},\mathtt{o})}{P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \frac{\partial P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})}{\partial \lambda} \qquad (3)$$

The partial derivative of $P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})$ w.r.t. λ can be computed as

$$\frac{P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})}{\partial \lambda}$$
$$= \mu(\mathtt{h} \mid \text{head}(\text{cl})\theta_{\mathtt{H}}, M) \cdot \mu(\mathtt{o} \mid \text{obs}(\text{cl})\theta_{\mathtt{H}}\theta_{\mathtt{O}}, M) \qquad (4)$$

Substituting (4) back into (3) yields

$$\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial \lambda}$$
$$= \sum_{\mathtt{b},\mathtt{h},\mathtt{o}} \Bigg(\frac{ec(\mathtt{b},\mathtt{h},\mathtt{o})}{P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \mu(\mathtt{h} \mid \text{head}(\text{cl})\theta_{\mathtt{H}}, M)$$
$$\cdot \mu(\mathtt{o} \mid \text{obs}(\text{cl})\theta_{\mathtt{H}}\theta_{\mathtt{O}}, M) \Bigg) \qquad (5)$$

The selection probability follows a naïve Bayes approach. Therefore, one can show in a similar way as for transition probabilities that

$$\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial \lambda}$$
$$= \sum_{\mathtt{b},\mathtt{h},\mathtt{o}} \Big(\frac{ec(\mathtt{b},\mathtt{h},\mathtt{o})}{P(\mathtt{h},\mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \sum_{\mathrm{cl}} c(\lambda, \mathrm{cl}, \mathtt{b}, \mathtt{h}, \mathtt{o}) \cdot$$
$$\cdot P(\mathrm{cl} \mid M, \boldsymbol{\lambda}) \cdot \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\theta_{\mathtt{H}}, M) \cdot$$
$$\cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\theta_{\mathtt{H}}\theta_{\mathtt{O}}, M) \Big) \quad (6)$$

where $c(\lambda, \mathrm{cl}, \mathtt{b}, \mathtt{h}, \mathtt{o})$ is the number of times that the domain element associated with λ is selected to ground cl w.r.t. h and o.

In the problem at hand, the described method has to be modified to take into account that $\lambda \in [0, 1]$ and that corresponding λ's sum to 1.0. A general solution, which we used in our experiments, is to reparameterize the problem so that the new parameters automatically respect the constraints no matter what their values are. To do so, we define the parameters $\boldsymbol{\beta}$ where $\beta_{ij} \in \mathbb{R}$ such that $\lambda_{ij} = \exp(\beta_{ij}) / (\sum_l \exp(\beta_{il}))$. This enforces the constraints given above, and a local maximum w.r.t. $\boldsymbol{\beta}$ is also a local maximum w.r.t. $\boldsymbol{\lambda}$, and vice versa. The gradient w.r.t the β_{ij}'s can be found by computing the gradient w.r.t. the λ_{ij}'s and then deriving the gradient w.r.t. $\boldsymbol{\beta}$ using the chain rule.

Discussion on SAGEM: What do we gain from SAGEM over the naïve approach? The expected ground counts $ec(\mathtt{b},\mathtt{h},\mathtt{o})$ are used as the sufficient statistics to evaluate all the neighbors. Evaluating neighbors is thus now independent of the number and length of the data cases—a feature which is important for scaling up. More precisely, the running time per neighbor evaluation is basically $\mathcal{O}(\#\textit{Gradient iterations} \cdot \#\textit{Ground transitions})$ because SAGEM's gradient approach does not perform LOHMM inferences.

The greedy approach is not always enough. For instance, if two hidden states are equivalent, to make them effectively differ from each other, one needs to make them differ both in visiting probabilities of the state and in behavior in the state, possibly requiring two steps for any positive effect. Fixing the expected counts in SAGEM worsens the problem, since changes in visiting probabilities of states do not show up before a LOHMM inference is made. To overcome this, different search strategies, such as beam search, can be used: Instead of a current hypothesis, a fixed-size set of current hypotheses is considered, and their common neighborhood is searched for the next set.

To summarize, SAGEM explores different abstraction levels due to ILP refinement operators, and, due to a GEM approach, it reduces the neighborhood evaluation problem to one that is solvable more efficiently.

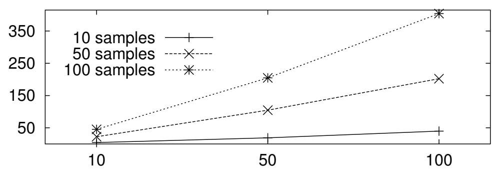


Figure 2: Speed-up (y axis), i.e., the ratio of time per EM iteration (in sec.) and time per SAGEM's gradient approach to evaluate neighbors. The speed-up is shown for different numbers of sequence lengths (x axis) and for different numbers of data cases (curves).

6 Experimental Evaluation

Our intentions here are to investigate whether SAGEM can be applied to real world domains. More precisely, we will investigate whether SAGEM

H1 speeds-up neighbor evaluation considerably (compared to the naïve learning algorithm);
H2 finds a comprehensible model;
H3 works in the presence of transition ambiguity;
H4 can be applied to real-world domains and is competitive with standard machine learning algorithms such as nearest-neighbor and decision-tree learners.

To this aim, we implemented the SAGEM using the Prolog system YAP-4.4.4. The experiments were run on a Pentium-III-2.3 GHz machine. For the improvement of expected score, we adapted the scaled conjugate gradient as implemented in Bishop and Nabney's Netlab library (`http://www.ncrg.aston.ac.uk/netlab/`) with a maximum number of 10 iterations and 5 random restarts.

Experiments with Synthetic Data: We sampled independently $10, 50, 100$ sequences of length $10, 50, 100$ (100 to 10000 ground atoms in total) from the LOHMM shown in Fig. 1**(a)** and computed their ground counts w.r.t. the samples. We measured the averaged running time in seconds per iteration for both, the naïve algorithm and SAGEM's gradient approach to evaluate neighbors when applied to the LOHMM shown in Fig. 1**(b)**. The times were measured using YAP's built-in `statistics/2`. The results are summarized in Fig. 2 showing the ratio of running times of naïve over SAGEM's gradient approach. In some cases the speed-up was more than 400. EM's lowest running time was 0.075 seconds (for 10 sequences of length 10). In contrast, SAGEM was constantly below 0.017 seconds. This suggests that **H1** holds.

We sampled 2000 sequences of length 15 (30000 ground atoms) from the LOHMM in Fig. 1**(a)**. There were 4 filenames, 2 users types. The initial hypothesis was the LOHMM in Fig. 1**(b)** with randomly initialized parameters. We run SAGEM on the sampled data. [3] Averaged

[3]The naïve algorithm was no longer used for comparison due

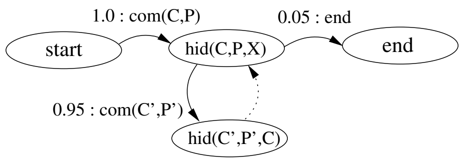


Figure 3: The initial hypothesis for the experiments with real-world data is a minimal structure, implying learning from scratch. C stands for command and P for parameters. The hidden state hid contains the new command, parameters and the latest old command.

over 5 runs, estimating the parameters for the initial hypothesis achieved a score of -47203. In contrast, the score of SAGEM's selected model was -26974 which was even slightly above the score of the original LOHMM (-30521). This suggests that **H3** holds. Moreover, in all runs, SAGEM included e.g. $\texttt{latex}(\texttt{A},\texttt{B}) \xleftarrow{0.61:\texttt{emacs(A)}} \texttt{emacs}(\texttt{A},\texttt{B})$ and $\texttt{emacs}(\texttt{A},\texttt{B}) \xleftarrow{0.48:\texttt{emacs(A)}} \texttt{latex}(\texttt{A},\texttt{B})$ which were not present in the initial model. This suggests that **H2** holds.

Experiments with Real-World Data: Finally, we applied SAGEM to the data set collected by Greenberg [8]. The data consists of 168 users of four groups: computer scientists, non-programmers, novices and others. About 300000 commands have been logged in on average 110 sessions per user. We present here results for two classes: novice-1(NV) consisting of 2512 ground atoms and non-programmers-4 (NP) consisting of 5183 ground atoms. We randomly selected 35 training sessions (about 1500 commands) for each class. On this data, we let SAGEM select a model for each class independently, starting from the initial hypothesis described in Fig. 3. To evaluate, we computed the plug-in estimates of each model for the remaining sessions corrected by the class priors. Averaged over five runs, the precision (0.94 ± 0.06 NV, 0.91 ± 0.02 NP) and recall values (0.67 ± 0.03 NV, 0.89 ± 0.05 NP) were balanced and the overall predictive accuracy was 0.92 ± 0.01. Jacobs and Blockeel [9] report that a kNN approach achieved a precision of 0.91 and J48 (WEKA's implementation of Quinlan's C4.5 decision tree learner) of 0.86 averaged over ten runs on 50 randomly sampled training examples. This suggests that **H4** holds. The used kNN and J48 methods, however, do not yield generative models and lack comprehensibility. SAGEM's selected models encoded e.g. "*non-programmers are very likely to type in* `cd..` *after performing* `ls` *in some directory*". This pattern was not present in the NV model. This suggests that **H2** holds.

7 Related Work

Statistical relational learning (SRL) can be viewed as combining ILP principles (such as refinement operators) with statistical learning, see [3] for an overview and references. Most attention, however, has been devoted to developing highly expressive formalisms. LOHMMs can be seen as an attempt towards *downgrading* such highly expressive frameworks. They retain most of the essential logical features but are easier to understand, adapt and learn. For the same reasons, simple statistical techniques (such as logistic regression or naïve Bayes) have been combined with ILP refinement oprators for traversing the search space, see e.g. [20, 14]. They, however, do not select dynamic models.

LOHMMs are related to Anderson *et al.*'s *relational Markov models* (RMMs) [1]. Here, states can be of different types, with each type described by a different set of variables. The domain of each variable is hierarchically structured. The main differences are that neither variable bindings, unification nor hidden states are supported. RMMs do not select the most-specific transition to resolve conflicting transitions. Instead, they interpolate between conflicting ones. This is an interesting option for LOHMMs because it makes parameter estimation more robust. On the other hand, it also seems to make it more difficult to adhere one of our design principles: locally interpretable transitions. Structure learning has been addressed based on *probability estimation trees*.

Logical sequences can be converted into binary trees by putting each instance of a relation symbol into a node. The left subtree represents the first argument and the right subtree represents the next atom in the list (of observations or arguments). Methods for learning tree languages [2] can thus be used for learning probabilistic models for logical sequences, too. The main differences, though, is that variable bindings are not supported.

LOHMMs are related to several extensions of HMMs such as factorial HMMs [7]. Here, state variables are decomposed into smaller units. The key difference to LOHMMs is that these approaches do not employ logical concepts.

Finally, SAGEM is related to more advanced HMM model selection methods. *Model merging* [22] starts with the most specific model consistent with the training data and generalizes by successively merging states. Abstract transitions, however, aim at good generalization, and the most general clauses can be considered to be the most informative ones. Therefore, *successive state splitting* [19] refines hidden states by splitting them into new states. In both cases, the authors are not aware of adaptions of Friedman's SEM.

8 Conclusions

A novel model selection method for logical hidden Markov models called SAGEM has been introduced. SAGEM combines *generalized EM*, which optimizes parameters, with structure search for model selection using ILP refinement operators. Experiments show SAGEM's effectiveness.

Future work should address other scores; other refinement

to unreasonable running times.

operators e.g. handling functors, deleting transitions, and generalizing hypotheses; logical pruning criteria for hypotheses; and efficient storing of ground counts. Moreover, the authors hope that the presented work will inspire further research at the intersection of ILP and HMMs.

Acknowledgments: The authors thank the three anonymous reviewers for helpful comments, Luc De Raedt for discussions on the topic, Nico Jacobs for discussions on user modeling, and Saul Greenberg for providing the data. The research was partially supported by the EU IST programme, contract no. FP6-508861, *APrIL II*, and by the Finnish Centre of Excellence Programme (2000-2005) under the project *New Information Processing Principles*. Tapani Raiko was partially supported by a Marie Curie fellowship at DAISY, HPMT-CT-2001-00251.

References

[1] C. R. Anderson, P. Domingos, and D. S. Weld. Relational Markov Models and their Application to Adaptive Web Navigation. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-02)*, 2002.

[2] R.C. Carrasco, J. Oncina, and J. Calera-Rubio. Stochastic inference of regular tree languages. *Machine Learning*, 44(1/2):185–197, 2001.

[3] L. De Raedt and K. Kersting. Probabilistic Logic Learning. *ACM-SIGKDD Explorations*, 5(1):31–48, 2003.

[4] L. De Raedt and K. Kersting. Probabilistic Inductive Logic Programming. In *Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT-2004)*, volume 3244 of *LNCS*, pages 19–36, Padova, Italy, October 2–5 2004. Springer.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc.*, B 39:1–39, 1977.

[6] N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-1997)*, 1997.

[7] Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.

[8] S. Greenberg. Using Unix: collected traces of 168 users. Technical report, Dept. of Computer Science, University of Calgary, Alberta, 1988.

[9] N. Jacobs and H. Blockeel. User modeling with sequential data. In *Proceedings of 10th International Conference on Human - Computer Interaction*, volume 4, pages 557–561, 2003.

[10] S. M. Katz. Estimation of probabilities from sparse data for hte language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing (ASSP)*, 35:400–401, 1987.

[11] K. Kersting, L. De Raedt, and T. Raiko. Logical hidden markov models. 2005. (submitted).

[12] K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden markov models. In *Proceedings of the Pacific Symposium on Biocomputing (PSB-03)*, 2003.

[13] K. Kersting, T. Raiko, and L. De Raedt. A Structural GEM for Learning Logical Hidden Markov Models. In *Working Notes of the Second KDD-Workshop on Multi-Relational Data Mining (MRDM-03)*, 2003.

[14] N. Landwehr, K. Kersting, and L. De Raedt. nFOIL: Integrating Naïve Bayes and Foil. In *Proceedings of AAAI-05*, 2005. To appear.

[15] T. Lane. Hidden Markov Models for Human/Computer Interface Modeling. In *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pages 35–44, Stockholm, Sweden, July 1999.

[16] G. J. McKachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Eiley & Sons, Inc., 1997.

[17] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.

[18] C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. Declarative Bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.

[19] M. Ostendorf and H. Singer. HMM topology design using maximum likelihood successive state splitting. *Computer Speech and Language*, 11(1):17–41, 1997.

[20] A. Popescul, L. H. Ungar, S. Lawrence, and D. M. Pennock. Statistical Relational Learning for Document Mining. In *Proceedings of ICDM-03*, pages 275–282, 2003.

[21] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–15, January 1986.

[22] A. Stolcke and S. Omohundro. Hidden Markov model induction by Bayesian model merging. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Adv. in Neural Information Processing Systems*, volume 5, pages 11–18, 1993.