Ma 108

ACTA POLYTECHNICA SCANDINAVICA

MATHEMATICS AND COMPUTING SERIES No. 108

Bayesian Ensemble Learning for Nonlinear Factor Analysis

HARRI VALPOLA

Helsinki University of Technology
Neural Networks Research Centre
P.O. Box 5400
FIN-02015 HUT
Finland

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission for public examination and debate in Auditorium T2 of Helsinki University of Technology (Espoo, Finland) on the 10th of November, 2000, at 12 o'clock noon.

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

ESPOO 2000

Valpola, H., **Bayesian Ensemble Learning for Nonlinear Factor Analysis**. Acta Polytechnica Scandinavica, Mathematics and Computing Series No. 108, Espoo 2000, 54 pp. Published by the Finnish Academies of Technology. ISBN 951-666-552-7. ISSN 1456-9418. UDC 519.237.7:517.988:681.327.12:159.953.

Keywords: Nonlinear factor analysis, unsupervised learning, latent variable models, generative models, Bayesian probability theory, ensemble learning, neural networks.

ABSTRACT

An active research topic in machine learning is the development of model structures which would be rich enough to represent relevant aspects of the observations but would still allow efficient learning and inference.

Linear factor analysis and related methods such as principal component analysis and independent component analysis are widely used feature extraction and data analysis techniques. They are computationally efficient but are restricted to linear models. Many natural phenomena are nonlinear and therefore several attempts have been made to generalise the model by relaxing the linearity assumption. The suggested approaches have suffered from overfitting and the computational complexity of many of the algorithms scales exponentially with respect to the number of factors, which makes the application of these methods to high dimensional factor spaces infeasible.

This thesis describes the development of a nonlinear extension of factor analysis. The learning algorithm is based on Bayesian probability theory and solves many of the problems related to overfitting. The unknown nonlinear generative mapping is modelled by a multi-layer perceptron network. The computational complexity of the algorithm scales quadratically with respect to the dimension of the factor space which makes it possible to use a significantly larger number of factors than with the previous algorithms. The feasibility of the algorithm is demonstrated in experiments with artificial and natural data sets. Extensions which combine the nonlinear model with non-Gaussian and dynamic models for the factors are introduced.

PREFACE

The work reported in this thesis was carried out in the Neural Networks Research Centre, Helsinki University of Technology, during 1996–2000. The work was mainly funded by the Helsinki Graduate School of Computer Science and Engineering. Additional support came from Jenny and Antti Wihuri Foundation.

I wish to thank Professors Juha Karhunen and Erkki Oja for supervising my work and Academy Professor Teuvo Kohonen for guiding the early steps of my scientific career and teaching me what scientific research is about.

I also wish to thank the co-authors of the publications included in this thesis, Xavier Giannakopoulos, James Miskin, Antti Honkela and Dr. Petteri Pajunen, as well as Juha Reunanen who did much of the preliminary experimental work for the last publication.

The manuscript of this thesis was reviewed by Professor Mark Girolami and Dr. Petri Myllymäki. I am greatful for their expert comments on the structure and scientific content which led to considerable improvements of the thesis. I also wish to express my sincere gratitude for the valuable comments and constructive criticism Professors Juha Karhunen and Erkki Oja gave on the earlier versions of the manuscript.

Special thanks go to the personnel of the Neural Networks Research Centre and the Laboratory of Computer and Information Science who have created a friendly and inspiring atmosphere in which to work.

I thank Lotta for her patience and tolerance during the time of writing the manuscript.

Espoo, October 2000

Harri Valpola

Harri Valpola

CONTENTS

LIST OF ABBREVIATIONS AND GLOSSARY OF TERMS

Abbreviations

EM	expectation maximisation
FA	factor analysis
GTM	generative topographic mapping
ICA	independent component analysis
IFA	independent factor analysis
MAP	maximum a posteriori
MDL	minimum description length
ML	maximum likelihood
MLP	multi-layer perceptron
PCA	principal component analysis
RBF	radial basis function
SOM	self-organising map
ST	signal transformation

Glossary of terms

artificial neural network: A model which consists of simple building-blocks. The development of such models has been inspired by neurobiological findings. The building-blocks are termed neurons in analogy to biological brain.

auto-associative learning: Representation of the observations is learned by finding a mapping from observations to themselves through an “information bottleneck” which forces the model to produce a compact coding of the observations. The **recognition model** and **generative model** are learned simultaneously.

Bayesian probability theory: In Bayesian probability theory, probability is a measure of subjective belief as opposed to frequentist statistics where probability is interpreted as the relative frequency of occurrences in an infinite sequence of trials.

factor: In **generative models**, the regularities in the observations are assumed to have been caused by underlying factors, also termed hidden causes, latent variables or sources.

factor analysis: A technique for finding a **generative model** which can represent some of the statistical structure of the observations. Usually refers to linear factor analysis where the generative model is linear.

feature: Feature describes a relevant aspect of observations. The term is often used in connection with **recognition models**. Bears resemblance to the term **factor** which is more common in connection to **generative models**.

ensemble learning: A technique for approximating the exact application of **Bayesian probability theory**.

generative model: A model which explicitly states how the observations are assumed to have been generated. See **recognition model**.

graphical model: A graphical representation of the causal structure of a probabilistic model. Variables are denoted by circles and arrows are used for representing the conditional dependences.

hidden cause: See **factor**.

latent variable: See **factor**.

posterior probability: Expresses the beliefs after making an observation. Sometimes referred to as the posterior.

prior probability: Expresses the beliefs before making an observation. Sometimes referred to as the prior.

probability density: Any single value of a continuous valued variable usually has zero probability and only a finite range of values has a nonzero probability. Probability of a continuous variable can be characterised by probability density which is defined to be the probability of a range divided by the size of the range.

probability mass: In analogy to physical mass and density, ordinary probability can be called probability mass in order to distinguish it from **probability density**.

recognition model: A model which states how **features** can be obtained from the observations. See **generative model**.

signal transformation approach: Finds a **recognition model** by optimising a given criterion over the resulting features.

source: See **factor**.

supervised learning: Aims at building a model which can mimic the responses of a "teacher" who provides two sets of observations: inputs and the corresponding desired outputs. See **unsupervised learning**.

unsupervise learning: The goal in unsupervised learning is to find an internal representation of the statistical structure of the observations. See **supervised learning**.

volume: In analogy to physical mass, density and volume, the size of range of continuous valued variables can be called volume. See **probability density**.

1 INTRODUCTION

Models are simplifications of reality which have the purpose of representing relevant aspects of a system under consideration whilst discarding minor details in order to reduce computational load. Models can be used for making predictions and inferences which are required for decision making and choosing actions.

In some cases there is well established theory which can be used for constructing the model. However, for applications such as speech recognition, computer vision and autonomous robotics, the theoretical knowledge is either incomplete or produces models which are too complex and detailed to be of any practical use. In such applications, machine learning has proven to be a successful approach to model building. This means that the learning system is given a flexible set of possible models from which it selects the ones that seem to explain the observations.

An active research topic in machine learning is the development of model structures which are rich enough to represent the relevant aspects of the observations but at the same time allow efficient learning and inference. This is also the topic of the present thesis.

Theoretically the richest model would be the universal Turing machine, which can represent anything that is programmable with computers [41]. The problem with this model is that the space of programmes has too complex a structure to aid in finding good representations for given observations. Therefore the use of this model is restricted to very simple problems where the richness of the model cannot be utilised.

At the other end of the spectrum are structurally very limited but computationally powerful models. The linear factor analysis model is a typical example. These models can be applied to vast data sets but the problem is that most of the interesting structure will remain hidden because the model has no means of representing it.

Artificial neural networks are models which are structurally very simple because they consist of very simple building blocks, neurons[1]. The term originates from models which were inspired by certain structures observed in the brain but in the current meaning the neural network models may not have anything to do with the biological brain. The linear factor analysis model, for instance, can be viewed as a very simple neural network. The idea is that from very elementary, computationally efficient building blocks it is possible to build models which have a rich representational capacity. The hope is that the resulting model will also be computationally efficient. The research is driven by the knowledge that the brain seems to have solved the problem: we are able to find regularities in our environment and learn abstractions which capture the structure of the regularities and allow us to predict future observations and plan our actions.

The goal of unsupervised learning is to extract an efficient representation of the statistical structure implicit in the observations [43]. Factor analysis is one example of unsupervised learning of latent variable models [25]. The observations are modelled as having been generated by unknown latent variables via an unknown mapping. Typically these models are learned by alternating between estimating the latent variables assuming that the model is fixed and estimating the model assuming that the latent variables are fixed.

The linear factor analysis model states that the observations are generated by a linear but otherwise unknown mapping from continuous valued latent variables, factors. The linearity assumption is restrictive and unrealistic in many cases and therefore several attempts have been made to relax the assumption. This extension has turned out to be very

[1]When referring to 'neurons', the computational neurobiological interpretation is assumed.

difficult and most existing methods are able to handle only models where the number of factors is quite low.

1.1 Contributions and structure of the thesis

This thesis consists of eight publications and an introductory part with literature survey. The aim of the thesis is to develop a computationally efficient algorithm for a nonlinear extension of the linear factor analysis model.

- The main result of this thesis is the development of an algorithm which is able to learn nonlinear factor analysis models with a relatively high number of factors. The computational complexity scales quadratically with respect to the dimension of factor space which is already efficient enough for many interesting applications, but it is also possible to use the model as an elementary module in larger models which scale linearly in the total size of the representation.
- Bayesian probability theory and decision theory provide the theoretical framework for learning, inference and decision making. In this thesis, a computationally feasible approximation of the exact Bayesian learning, termed ensemble learning, is used. The manner in which ensemble learning can be applied to nonlinear factor analysis is presented.
- It is shown how the nonlinear model can be combined with other extensions of factor analysis which relax the Gaussianity assumption of the factors and include a model for the dynamics of the factors.

Section 2 of the introductory part summarises the publications of the thesis, with the contributions of the author explained. Section 3 outlines the theoretical framework of Bayesian probability theory and decision theory. Practical methods and approximations together with their connection to information theory are discussed in section 4. Section 5 introduces the basic static Gaussian linear factor analysis model and its non-Gaussian, nonlinear and dynamic extensions. Publication V serves as a detailed account on the nonlinear factor analysis method developed in this thesis but section 6 gives a brief summary. Biological relevance and further lines of research are discussed in section 7.

2 PUBLICATIONS OF THE THESIS

2.1 Original papers

The following eight publications describe the development of methods suitable for unsupervised learning of nonlinear factor analysis models.

I H. Lappalainen[2], "Using an MDL-based cost function with neural networks," in *Proceedings of the International Joint Conference on Neural Networks, IJCNN'98*, (Anchorage, Alaska, USA, May 4–9), pp. 2384–2389, 1998.

II H. Lappalainen, "Ensemble Learning for Independent Component Analysis," in *Proceedings of the First International Workshop on Independent Component Analysis and Blind Signal Separation, ICA'99*, (Aussois, France, Jan. 11–15), pp. 7–12, 1999.

III H. Lappalainen and X. Giannakopoulos, "Multi-Layer Perceptrons as Nonlinear Generative Models for Unsupervised Learning: a Bayesian Treatment," in *Proceedings of the Ninth International Conference on Artificial Neural Networks, ICANN'99*, (Edinburgh, UK, Sep. 7–10), pp. 19–24, 1999.

IV H. Lappalainen and J. W. Miskin, "Ensemble Learning," in *Advances in Independent Component Analysis* (M. Girolami, ed.), pp. 76–92, Springer-Verlag, 2000.

V 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, Springer-Verlag, 2000.

VI H. Valpola, "Nonlinear Independent Component Analysis Using Ensemble Learning: Theory," in *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, (Helsinki, Finland, June 19–22), pp. 251–256, 2000.

VII H. Valpola and P. Pajunen, "Fast Algorithms for Bayesian Independent Component Analysis," in *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, (Helsinki, Finland, June 19–22), pp. 233–237, 2000.

VIII H. Valpola, "Unsupervised learning of nonlinear dynamic state-space models," Publications in Computer and Information Science A59, Helsinki University of Technology, Espoo, Finland, 2000.

2.2 Contents of the publications and author's contributions

Publication I lays the foundation of the thesis. A cost function for multi-layer perceptron (MLP) networks is developed in the minimum description length (MDL) framework. The presentation is given from the point of view of supervised learning as this is the learning method traditionally used with MLP networks. The possibility of unsupervised learning is briefly outlined. It transpired that the cost function has a Bayesian interpretation and essentially the same approach works for ensemble learning, the Bayesian method used in the remainder of the publications.

[2]The former name of the present author was Lappalainen.

Publication II demonstrates that it is possible to apply ensemble learning to the unsupervised learning of linear independent component (or factor) analysis. The emphasis is on using full posterior approximations instead of point estimates, as that allows comparing between models and protects against overfitting. The treatment of the posterior distribution of variables which have mixture-of-Gaussians prior distributions is later replaced by the method borrowed from [3], but the treatment of other distributions in later publications is based on the methods presented here.

In publication III, multi-layer perceptrons are used for nonlinear factor analysis following the guidelines presented in publications I and II. The results are encouraging although the model used is rather small. The strategy for growing the network is inspired by the biological brain. Mr. Xavier Giannakopoulos assisted in running the simulations.

Publication IV is a tutorial introduction to ensemble learning and appeared in the same volume as publication V. It is co-authored by Mr. James Miskin, who also had another paper using ensemble learning in the same volume. The credit of writing the part considering free form approximation in ensemble learning should go to James Miskin. The present author was responsible for the rest of the text: fixed form approximation, model selection and the relation to coding and the EM algorithm.

Most of the results concerning nonlinear factor analysis by an MLP network are contained in publication V. The article also includes a detailed account on the learning scheme used in the simulations. It turned out that the simple approximation developed in publication I and applied in publication III does not suffice for unsupervised learning of MLP networks. A more accurate and laborious approximation is used in publication V. The article also outlines nonlinear independent factor analysis which results as the combination of a nonlinear mapping and a non-Gaussian model for factors. Mr. Antti Honkela assisted in running the simulations.

Publication VI presents a more accurate and detailed derivation for nonlinear independent factor analysis than that provided in publication V.

The basic idea on which publication VII is based was first published in the technical report [72] which provides a new interpretation for the FastICA algorithm. This interpretation suggests various extensions of which a fast Bayesian independent component analysis algorithm utilising ensemble learning is given as an example. Dr. Petteri Pajunen contributed to writing and clarifying the style of presentation.

Publication VIII outlines a dynamic extension of the nonlinear independent factor analysis algorithm. The dynamics of the factors, or states in this case, are modelled by the same principles as the mapping from factors to observations. The extension is simple but has practical significance since time sequences with significant time structure are often encountered in practical problems.

3 BAYESIAN PROBABILITY THEORY

The starting point in modern Bayesian probability theory is that probability is interpreted as a *degree of belief* (for bibliographic notes, see [7, 57]). Richard Cox showed that certain very general requirements for the calculus of beliefs result in the rules of probability theory [19]. Decision theory also leads to the same rules [129, 114, 102] with the same interpretation. There are other domains, most notably measure theory, where the same rules appear, but from the point of view of learning systems and decisions in the face of uncertainty, degree of belief is the appropriate interpretation.

Beliefs are always subjective, and therefore all the probabilities appearing in Bayesian probability theory are conditional. In particular, under the belief interpretation probability is not an objective property of some physical setting, but is conditional to the prior assumptions and experience of the learning system. It is completely reasonable to talk about "the probability that there is a tenth planet in the solar system" although this planet either exists or does not exist and there is no sense in interpreting the probability as a frequency of observing a tenth planet. Sometimes the probabilities can be roughly equated with empirical frequencies, but this can be considered as a special case of the belief interpretation as was shown by Cox [19].

Accessible introductions to practical applications of Bayesian probability theory can be found, for instance, in [75, 103, 28].

3.1 Propositions

In Bayesian probability theory, probabilities are defined for propositions which follow the laws of Boolean algebra [11, 132] and can be either true or false, i.e., propositions which satisfy the laws of ordinary logic. We shall denote the propositions by capital letters. Three basic operations are defined in Boolean algebra: conjunction, disjunction and negation. We shall denote "A and B" by AB, "A or B" by $A + B$ and "not A" by $\neg A$.

Natural language involves propositions whose truth value is ambiguous, "sky is blue" for example. The definitions of words sky and blue are more or less ambiguous and therefore it is possible to think that the truth value of "sky is blue" is neither completely true nor completely false but something in between. Fuzzy logic [136, 133] tries to capture the ambiguity of propositions in natural language, but we shall consider only unambiguously defined propositions[3].

3.2 Elementary rules of Bayesian probability theory

Bayesian probability theory can be conveniently summarised in the following elementary rules:

Sum rule: $P(A|B) + P(\neg A|B) = 1$

Product rule: $P(AB|C) = P(A|C)P(B|AC)$

[3]Since fuzzy logic is sometimes presented as an alternative to Bayesian probability theory, it should be emphasised that truth value and degree of belief are different dimensions. It is possible to envision an extension of Bayesian probability theory which would define probabilities for propositions with fuzzy truth values.

Here $P(A|B)$ denotes the probability of A on the condition that B is true. These rules correspond to the negation and conjunction operations of Boolean algebra. The disjunction does not need a separate rule because it can be derived from negation and conjunction: $A + B = \neg(\neg A \neg B)$. In fact, only one operation would suffice since other operations can be derived from either NAND or NOR operation alone. The NAND operation, for instance, yields the following rule, starting from which every other rule of Bayesian probability theory can be derived: $P(\neg A + \neg B|C) + P(A|C)P(B|AC) = 1$.

These rules fix the scale on which the degrees of belief are measured. Cox showed that under very general requirements of consistency and compatibility with common sense, the rules of calculus with beliefs have to be homomorphic with the sum and product rule [19]. This means that one can measure the degrees of beliefs on any scale, but it is possible to transform the degrees of beliefs on the canonical scale of probabilities such that the rules for negation and conjunction take the form of the sum and product rule.

3.2.1 Bayes' rule

From the product rule it is possible to derive Bayes' rule:

$$P(A|BC) = P(A|C)P(B|AC)/P(B|C)\,. \tag{1}$$

If we assume that A is one of several explanations for the new observation, B is the new observation and C summarises all prior assumptions and experience, we notice that Bayes' rule tells how the learning system should update its beliefs as it receives a new observation.

Before making the observation B, the learning system knows only C, but afterwards it knows BC, that is, it knows "B and C". Bayes' rule then tells how the learning system should adapt $P(A|C)$ into $P(A|BC)$ in response to the observation. In order for Bayes' rule to be useful, the explanation A needs to be such that together with the prior assumptions and experience C it fixes the probability $P(B|AC)$.

Usually $P(A|C)$ is called the prior probability and $P(A|BC)$ the posterior probability. It should be noted, however, that this distinction is relative to the observation; the posterior probability for one observation is the prior probability for the next observation.

3.2.2 Marginalisation principle

While Bayes' rule specifies how the learning system should update its beliefs as new data arrives, the marginalisation principle provides for the derivation of probabilities of new propositions given existing probabilities. This is useful for prediction and inference.

Suppose the situation is the same as in the example with Bayes' rule, but now the learning system tries to compute the probability of making observation B before it has actually made the observation, that is, the learning system tries to predict the new observation.

Suppose $A_1, A_2, \ldots$ are exhaustive and mutually exclusive propositions, in other words, exactly one of A_i is true while the rest are false. As before, assume that A_i are possible explanations for B and the prior assumptions and experience C are such that both $P(B|A_iC)$ and $P(A_i|C)$ are determined. The marginalisation principle then states the following:

$$P(B|C) = \sum_i P(A_i|C)P(B|A_iC)\,. \tag{2}$$

The probability of B thus depends on the prior probabilities $P(A_i|C)$ of the different explanations and the probability $P(B|A_iC)$ which each explanation gives to B.

Notice also that $P(B|C)$ appears in Bayes' rule, but the marginalisation principle shows that it can be computed from $P(A_i|C)$ and $P(B|A_iC)$ alone. Therefore $P(A_i|C)$ and $P(B|A_iC)$ suffice for computing the posterior probability $P(A_i|BC)$:

$$P(A_i|BC) = \frac{P(A_i|C)P(B|A_iC)}{\sum_j P(A_j|C)P(B|A_jC)}\,. \tag{3}$$

3.3 Decision theory

Beliefs alone are not sufficient for making decisions. Preferences are also needed. Decision theory points out how the beliefs and preferences should be combined when making decisions. We shall denote by $U(A)$ the utility of proposition A. By definition, A is preferred over B if $U(A) > U(B)$.

Decision theory can be summarised in a single rule:

$$U(A) = P(B|A)U(AB) + P(\neg B|A)U(A\neg B)\,. \tag{4}$$

We shall call it the rule of expected utility. In case of mutually exclusive and exhaustive propositions $B_1, B_2, \ldots$, it generalises into

$$U(A) = \sum_i P(B_i|A)U(AB_i)\,. \tag{5}$$

The significance of the rule becomes apparent if one considers A to be an action and B_i to be the possible consequences. Basically the rule indicates that the utility of A depends on the utilities of the possible consequences of A, weighted by the probabilities of the consequences.

The sum and product rules of probability theory fix the scale by which degrees of beliefs are measured to be the canonical scale of probabilities. The rule of expected utility does the same for utilities, up to linear scaling and an additive constant. This is because the beliefs have clear limits in absolute belief and disbelief while there are no absolutely worst or best possible states of the world, or if there are, the difference of their utility is probably infinitely greater than the difference between any other states of the world.

3.4 Summary: learning, reasoning and action

We can now summarise what Bayesian probability theory and decision theory say about learning, reasoning and action by giving a simple example. Suppose there are prior assumptions and experience I and possible explanations expressed as states of the world S_i. An observation D is made and an action A_j is chosen based on the belief about what is the consequence D' of the action. We assume D' is one of several possible observations D'_k made after the action is chosen.

The prior assumptions and experience I are assumed to be such that it is possible to determine the prior probability $P(S_i|I)$ of each state of the world; the probability $P(D|S_iI)$ of observation D given the state of the world S_i; the probabilities $P(D'_k|S_iA_jDI)$ of different consequences of actions given the state of the world and prior experience; and the utility of the consequences $U(A_jD'_kDI)$. The action A_j is assumed to have no effect on the state S_i of the world and thus $P(S_i|A_jDI) = P(S_i|DI)$.

The first stage of the example is learning. First the states of the world have prior probabilities $P(S_i|I)$. After making the observation D, the probabilities change according to Bayes' rule:

$$P(S_i|DI) = \frac{P(S_i|I)P(D|S_iI)}{\sum_{i'} P(S_{i'}|I)P(D|S_{i'}I)}. \tag{6}$$

The belief in those states of the world which were able to predict the observation better than average increases, and vice versa.

The next stage is to infer which consequences different actions have. According to the marginalisation principle,

$$P(D'_k|A_jDI) = \sum_i P(S_i|A_jDI)P(D'_k|S_iA_jDI). \tag{7}$$

Notice that A_j was assumed to have no effect on S_i and thus $P(S_i|A_jDI)$ is equal to the posterior probability $P(S_i|DI)$ which was computed in the first stage.

The third stage of the example is choosing an action which has the greatest utility. The utilities can be computed by the rule of expected utility:

$$U(A_jDI) = \sum_k P(D'_k|A_jDI)U(A_jD'_kDI). \tag{8}$$

The utilities of actions are based on the utilities of consequences and the probabilities of consequences in light of the experience, which were computed in the previous stage.

So far we have explicitly denoted that the probabilities are conditional to the prior assumptions and experience I. In most cases the context will make it clear which are the prior assumptions and usually I is left out. This means that probability statements like $P(S_i)$ should be understood to mean $P(S_i|I)$ where I denotes the assumptions appropriate for the context.

4 BAYESIAN LEARNING IN PRACTICE

The previous section outlined how learning, reasoning and decision making should function in theory. The theory does not take into account the computational and storage capacity requirements, however. In realistic situations exact computation following Bayes' rule, the marginalisation principle and the rule of expected utility is almost always computationally prohibitive. Therefore an important research topic has long been the development of methods which yield practical approximations of the application of the exact theory.

Currently the frequentist interpretation is the prevailing statistical philosophy. According to this view, probability measures the relative frequencies of different outcomes in an (imaginary) infinite sequence of trials. Probability is defined for random variables which can take different values in different trials. Both the frequency and belief interpretations have coexisted since the early days of probability theory (see, e.g., [75]). The advent of quantum physics gave the frequentist view strong impetus because probability was seen as a measurable property of the physical world.

The Bayesian and frequentist schools use different language and methods. In frequentist statistics, a hypothesis or a parameter of a model cannot have probabilities as they are not random variables and do not take different values in different trials. The methods developed within the frequentist school include estimators and confidence intervals for parameters and P-values for hypothesis testing, whereas prior and posterior probabilities do not enter the calculations.

In Bayesian statistics, probability measures the degree of belief and it is therefore admissible to talk about the probabilities of hypotheses and parameters. The probability distribution of a parameter, for instance, quantifies the uncertainty of its value. Many Bayesian statisticians avoid talking about random variables altogether because almost always one is actually talking about uncertainty of the outcomes of experiments, not some intrinsic property of the world.

The prevailing frequentist language and methods have caused misconceptions about the Bayesian view and not all researchers are ready to consider the Bayesian approach to statistical inference as theoretically optimal. Here are answers to some common arguments against the Bayesian approach to learning:

- "Prior probabilities are needed in Bayesian learning whereas other methods do not need them."

  Learning cannot start from a vacuum and therefore prior assumptions of some sort are needed. In some learning algorithms these assumptions are implicit but they still exist. In Bayesian learning, these assumptions are required to be stated explicitly, which makes it easier to locate possible flaws in them.

- "Bayesian learning works only if the true model is included in the hypothesis space."

  True models exist only in theoretical constructs. Bayesian statistics does not take a stand on true models because it only talks about the beliefs in propositions. In practice the prior knowledge about models almost never states that one of the models is true but that the models give better and worse predictions about the observations. Then the posterior probability does not measure the belief that a certain hypothesis is true but the belief that the observations can be best predicted by the hypothesis.

- "Various efficient learning methods are not derived from Bayesian probability theory but they still work."

  It is true that it is possible to develop efficient learning algorithms without referring to Bayesian probability theory. Evolution of the brain, for instance, was not guided by any theory for certain. However, this does not mean that Bayesian probability theory would not be an appropriate way to try to understand the learning algorithms. In practice they can be interpreted as approximations to the theoretically optimal exact Bayesian approach. It is then often easier to investigate their underlying assumptions and limitations and in some cases generalise the methods.

4.1 Probability density for real valued variables

In symbolic representations, the propositions are discrete and similar to simple statements of natural language. When trying to learn models of the environment, the problem with discrete propositions is that an unimaginable number of them is needed for covering all the possible states of the world. The alternative is to build models which have real valued variables. This allows one to manipulate a vast number of elementary propositions by manipulating real valued functions, probability densities.

Following the usual Bayesian convention, probability density is denoted by a lower case p and the ordinary probability by a capital P throughout this thesis. We also use the convenient short hand notation where $p(x|y)$ means the distribution of the belief in the value of x given y. Alternative notation would be $f_{X|Y}(x|y)$, which makes explicit the fact that $p(x|y)$ is not the same function as, for instance, $p(u|v)$. In cases where the ordinary probability needs to be distinguished from probability density, it is called probability mass in analogy to physical mass and density.

Bayes' rule looks exactly the same for probability densities as it does for probability mass. If a and b are real valued variables, Bayes' rule takes the following form:

$$p(a|bC) = \frac{p(a|C)p(b|aC)}{p(b|C)}. \tag{9}$$

This is convenient but also dangerous. It is all too easy to talk about "the single most probable model" when one is actually talking about the model which has the highest probability density. This is dangerous since probability density is a derived quantity and has no role *per se* in probability theory. This will be more evident when looking at the marginalisation principle

$$p(b|C) = \int p(a|C)p(b|aC)da \tag{10}$$

or the rule of expected utility

$$U(A) = \int p(b|A)U(Ab)db \tag{11}$$

written for probability densities. Notice that for probability density the sum changes into an integral. In the integrals, the impact on the probability $p(b|C)$ or on the utility $U(A)$ is zero at any single point if the density is finite. Only a nonzero range has a nonzero contribution in the integrals. It is then evident that a high density *per se* is not important, but the overall probability mass in the vicinity of a model is.

4.2 Methods for approximating the posterior probability

In Bayesian learning, the learning system updates its prior probability of models (explanations, states of the world, etc.) into posterior probability according to Bayes' rule. The updated probability can then be used for prediction or making decisions. In both cases, the computation involves a sum weighted by the posterior probability (or an integral in case of real valued parameters). The straight-forward numerical summation or integration is usually computationally far too expensive, and therefore various techniques have been developed for approximating the result.

Basically there are two complementing ways to reduce the required computation. One is to design the models so that the posterior probability will have a mathematically tractable, simple functional form. The other is to approximate the weighted sum or integral. The methods are complementary because the accuracy of the approximation depends on the complexity of the posterior probability which can be affected by the design of the model.

If it is known in advance which prediction or decision is going to be made based on the posterior probability, the approximation can and should take this into account. In many cases this information is not available, however, and then the best thing to do is to try to approximate those parts of the posterior probability which have the highest probability mass because those are the ones which have the strongest impact on the predictions and decisions.

4.2.1 Point estimates

The most efficient and least accurate approximation is, in general, a point estimate of the posterior probability. It means that only the model with highest probability or probability density is used for making the predictions and decisions. Whether the accuracy is good depends on how large a part of the probability mass is occupied by models which are similar to the most probable model.

The two point estimates in wide use are the maximum likelihood (ML) and the maximum a posteriori (MAP) estimator. The ML estimator neglects the prior probability of the models and maximises only the probability which the model gives for the observation. The MAP estimator chooses the model which has the highest posterior probability mass or density.

One should be particularly careful when using the MAP estimator with probability densities. The MAP estimator is useful in cases where the second order curvature of the logarithm of the posterior probability density with respect to the model parameters is roughly constant for all models. Then the widths of the peaks of the posterior density are roughly equal and probability mass around any of the models is proportional to the probability density of the model. The second order curvature can be affected by the parameterisation of the model, but in general it depends also on the observations.

For the real valued latent variable models considered in this thesis, the MAP estimators cannot be used as such. The models include products of unknown quantities, weights and factors in this case, which means that by increasing the value of one variable, the value of another variable can be decreased. This scaling does not change the model but the density of the first variable decreases and the second value increases. For each observation, a new set of values is estimated for factors which means that typically the number of unknown values for factors is far greater than the number of unknown values for weights. If the MAP estimates were used for factor analysis models, the result would be that the weights of the model would grow and the values of the factors shrink. The resulting low density of the weights would be overwhelmed by the high density of the factors. In other words, MAP estimation would find the values of the weights which give the highest density for the factors, but would not say much about the posterior probability mass of the model because the high density would be obtained at the cost of narrow posterior peaks of the factors.

In any case, the use of a point estimate will cause a phenomenon called overfitting. Most people are familiar with the concept at least in the context of fitting polynomials to observations. Using only the best model means being excessively confident that the best fit is the correct one. In the case of probability densities, for instance, *all* probability mass is in models which have a poorer fit than the best model. This means that the optimal prediction based on the full posterior density will necessarily be less confident about the fit than a prediction based on only the "best" model.

EM algorithm. The expectation-maximisation (EM) algorithm [21] is often used for learning latent variable models, including the factor analysis model [110]. It is a mixture of point estimation and analytic integration over posterior density. The EM algorithm is useful for latent variable models if the posterior probability of the latent variables can be computed when other parameters of the model are assumed to be known.

The EM algorithm was developed for maximum likelihood parameter estimation from incomplete data. Let us denote the measured data by x, the missing data by y and the parameters by θ. The algorithm starts with an estimate $\hat{\theta}_0$ and alternates between two steps, called E-step for expectation and M-step for maximisation. In the former, the conditional probability distribution $p(y|\hat{\theta}_i, x)$ of the missing data is computed given the current estimate $\hat{\theta}_i$ of the parameters and in the latter, a new estimate $\hat{\theta}_{i+1}$ of the parameters is computed by maximising the expectation of $\ln p(x, y|\theta)$ over the distribution computed in the E-step.

It can be proven that this iteration either increases the probability $p(x|\theta)$ or leaves it unchanged. The usefulness of the method is due to the fact that it is often easier to integrate the logarithmic probability $\ln p(x, y|\theta)$ than probability $p(x, y|\theta)$ which would be required if $p(x|\theta)$ were maximised directly.

The EM algorithm applies to latent variable models when the latent variables are assumed to be the missing data. When compared to simple point estimation, the benefit of the method is that fewer unknown variables are assigned a point estimate, thus alleviating the problems related to overfitting.

4.2.2 Stochastic sampling

In stochastic sampling one generates a set of samples of models, whose distribution approximates the posterior probability of the models [33]. There are several techniques having slightly different properties, but in general the methods yield good approximations of the posterior probability of the models but are computationally demanding. To some extent the trade-off between efficiency and accuracy can be controlled by adjusting the number of generated samples.

For simple problems, the stochastic sampling approach is attractive because it poses the minimal amount of restrictions on the structure of the model and does not require careful design of the learning algorithm. For an accessible presentation of stochastic sampling methods from the point of view of neural networks, see [92].

4.2.3 Parametric approximations

Parametric approximations lie in between point estimates and stochastic sampling in terms of computational complexity and accuracy of the approximation. The key idea is to replace the complex posterior probability by a simpler, mathematically tractable approximation.

A standard procedure for approximating the posterior density with a parametric posterior density is Laplace's method [71], where the logarithm of the posterior density is approximated by its Taylor series expansion around the maximum point, i.e., the MAP estimate. The most used is the second order expansion, which amounts to approximating the posterior density by the Gaussian distribution. The choice is done primarily for mathematical tractability, although it has been shown that under very general conditions the posterior density will approach the Gaussian distribution as the number of measurements grows. For a textbook account on Laplace's method, asymptotic normality of the posterior density and statistics in general, see e.g. [115].

Laplace's method, when applied to complex models, can suffer from the same problems as MAP estimation in general. If the MAP estimate fails to locate a point in parameter space which not only has high probability density but also is surrounded by large probability mass, the second order Taylor series expansion can recognise this, but cannot, in practice, guide the search for a better point estimate to start with because it would be computationally too expensive. Whether this is a problem depends on the models at hand. In supervised learning with neural networks, MacKay has obtained good results [81], but for unsupervised learning of complex models, the MAP estimate causes problems.

4.3 Information-theoretic approaches to learning

Information theory can offer a simple, intuitive point of view to learning. If we succeed in finding a very simple description for the observations, the argument goes, then we must have found interesting structure in the data. The description length and probability are tightly linked. According to Shannon's coding theory, the shortest expected description length for a proposition equals the negative logarithm of the probability of the proposition. Viewed like this, the information-theoretic approach to learning is nothing else than using a different scale for measuring the beliefs, and any learning method derived in the information-theoretic context can be readily translated into the Bayesian context by a simple transformation of scale. This is not to say that information theory did not have an independent justification in coding theory.

Concepts from the Bayesian framework often have intuitive interpretations in the coding context. The prior probabilities, for instance, translate into the specification of a coding scheme. Optimal encoding of an observation into a binary string produces a seemingly random string of ones and zeros and some prior knowledge is needed about the instructions for decoding. This corresponds to the Bayesian prior. Another example is that slight approximations to the exact Bayesian learning translate into slightly nonoptimal coding schemes. Ensemble learning is an example of a Bayesian approximation scheme whose roots are in coding schemes.

It may be that much of the success of approximation schemes first derived in the information-theoretic context is due to the widespread misuse of probability densities and MAP estimates. In coding context, it is easier to see that it is the probability mass that matters, not probability density, because it is clear that in order to measure the number of bits needed for coding a real valued variable, the precision of coding has to be specified. This corresponds to specifying the volume around a point in space and thus determines a probability mass.

4.3.1 Coding and complexity

Information theory studies communication and the information content of random variables. Its foundations were laid by Claude Shannon who studied communication over noisy channels and was able to show that the information content of observing a discrete random variable X is $L(X) = -\log_2 P(X)$ bits [117]. According to Shannon's coding theory, this is the number of bits needed for coding entities using optimal coding. Later Rissanen and Langdon developed arithmetic coding which is a practical algorithm for attaining a coding which is arbitrarily close to optimal [108, 109].

Solomonoff [119, 120], Kolmogorov [67] and Chaitin [15] independently developed algorithmic information theory which is based on the idea that the complexity of a binary string is equated with the shortest computer programme which produces that string. This measure, known as Kolmogorov complexity, is not computable because of the so called halting problem [41]. By taking into account the time it takes to run a programme, Kolmogorov complexity can be modified so that it becomes computable. This is known as Levin complexity [77]. Good introductions to coding and complexity can be found, for instance, in [78, 18].

In principle Levin complexity can be used for learning as demonstrated in [116]. However, the model space which includes all computer programmes lacks the structure which would help in developing efficient learning algorithms. This means that although applicable in principle, the method cannot learn complex models in practice.

From a Bayesian point of view, a complexity measure can be interpreted as a prior over binary strings. By using Shannon's formula in the other direction, the lengths of the computer programmes C can be interpreted to define a prior over programmes as $P(C) = 2^{-L(C)}$. Each programme gives a probability $P(S|C) = 1$ for the string S that the programme C generates. It is then clear that when n first bits S_n of a string are observed, it is possible to use the model to predict the continuation for the string using Bayes' rule and the marginalisation principle.

The marginalisation principle tells that the prediction should use all programmes C weighted by their posterior probabilities. However, often the development originating from coding tradition assumes using only the shortest programme C. An example of this is the stochastic complexity developed by Rissanen [107]. It replaces the computer programmes by ordinary parameterised statistical models in measuring the complexity. With simple models, at least, this leads to practical algorithms for finding the "shortest" model which gives the observation. Stochastic complexity extends ML estimation so that different model structures can be dealt with, but like ML estimation, it yields only point estimates.

4.3.2 Minimum message length inference

A different line of research, which leads to ensemble learning, is represented by [130, 106, 131, 44]. Like stochastic complexity, it also operates on parameterised statistical models rather than computer programmes.

We shall illustrate the treatment of real valued parameters by considering a simple model with one parameter θ which determines the probability for the observation x. The probabilities $p(x|\theta)$ and $p(\theta)$ are assumed to be specified. Then we ask how, using the given model, we can encode x into a message using the least amount of bits. Assume that we require x to be encoded with the precision ϵ_x. The purpose is not actually to encode x or send a message to someone, but to learn about θ by imagining a coding scheme.

Wallace and Boulton [130] suggested the following two-part message: first encode θ discretised with precision ϵ_θ using the prior probability $p(\theta)$ and then encode x using the model $p(x|\theta)$ and the encoded θ. This will produce a code with length $L(\theta) + L(x|\theta)$. If ϵ_θ is small, the number of bits used for the first part of the message is approximately

$$L(\theta) \approx -\log[p(\theta)\epsilon_\theta]. \tag{12}$$

The number of bits used for the second part of the message depends on the discretised value of θ. Assuming the target value for θ was θ_0, the discretised value lies between $\theta_0 - \epsilon_\theta/2$ and $\theta_0 + \epsilon_\theta/2$ with roughly uniform probability. This means that the expected number of bits used for the second part is approximately

$$L(x|\theta) = \int_{\theta_0-\epsilon_\theta/2}^{\theta_0+\epsilon_\theta/2} -\log[p(x|\theta)\epsilon_x]d\theta. \tag{13}$$

If ϵ_θ is small, it is possible to approximate the length of the second part of the message by using a second order Taylor series expansion of $-\log p(x|\theta)$ about θ_0.

Given x, the total message length is a function of θ_0 and ϵ_θ. The result of learning is the optimal value for both, which minimises the message length. Looking at the equations for $L(\theta)$ and $L(x|\theta)$, it is evident that there is an optimal value for ϵ_θ, because increasing the size of the discretisation bins will decrease $L(\theta)$ due to the term $-\log \epsilon_\theta$, but it will increase the term $L(x|\theta)$ because the discretisation errors will increase and the expected deviations from optimal θ_0 grow larger.

The optimal value for ϵ_θ depends on how quickly $p(x|\theta)$ drops as θ is moved further away from the optimal value, and it turns out that the optimal ϵ_θ is linearly dependent on the width of the maximum peak of $p(x|\theta)$. Therefore the optimal θ_0 will tell the most plausible value for the parameter, and ϵ_θ will tell how uncertain the value is.

An accessible introduction to learning based on this coding scheme, known as minimum message length inference, can be found in [97, 96, 6].

4.4 Ensemble learning

Ensemble learning is a technique for parametric approximation of the posterior probability where fitting the parametric approximation to the actual posterior probability is achieved by minimising their misfit. The misfit is measured with Kullback-Leibler information [70], also known as relative or cross entropy. It is a measure suited for comparing probability distributions and, more importantly, it can be computed efficiently in practice if the approximation is chosen to be simple enough.

The Kullback-Leibler information between two probability density functions $q(x)$ and $p(x)$ is

$$I_{KL}(q(x)||p(x)) = E_q\left\{\ln\frac{q(x)}{p(x)}\right\} = \int q(x)\ln\frac{q(x)}{p(x)}dx. \tag{14}$$

It has the following interpretation: suppose we are picking samples from distribution $q(x)$, Kullback-Leibler information then measures the average amount of information the samples give for deciding that the samples are not from distribution $p(x)$. If $q(x)$ and $p(x)$ are the same, then the amount of information is zero. On the other hand, if $q(x)$ gives finite probability mass to samples for which $p(x)$ gives zero probability, then a single such sample will reveal that the samples are not taken from $p(x)$ and the average information is infinite.

Regarding the approximation of posterior probability, the most important benefit of ensemble learning is that Kullback-Leibler information is sensitive to probability mass and therefore the search for good models focuses on the models which have large probability mass as opposed to probability density. The drawback is that in order for ensemble learning to be computationally efficient, the approximation of the posterior needs to have a simple factorial structure. This means that most dependences between various parameters cannot be estimated. On the other hand, it should be possible to use ensemble learning instead of MAP estimation as the first stage in Laplace's method.

In the present form, the method was first presented by Hinton and van Camp [44] and the name ensemble learning was given by MacKay in [82]. Ensemble learning can also be seen as a variational method [60] and it also has a connection to the EM algorithm [93].

4.4.1 Cost function

Publication IV discusses ensemble learning at length, but this section describes briefly the cost function used in ensemble learning. Let us denote the vector of all the unknown variables of the model by $\boldsymbol{\theta}$ and the vector of observations by $\mathbf{x}$ and suppose that the probabilities $p(\mathbf{x}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$ are defined. According to Bayes' rule, the posterior probability $p(\theta|x)$ of the unknown variables is

$$p(\boldsymbol{\theta}|\mathbf{x}) = \frac{p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{x})} \tag{15}$$

and the Kullback-Leibler information between the true posterior $p(\boldsymbol{\theta}|\mathbf{x})$ and its approximation $q(\boldsymbol{\theta}|\mathbf{x})$ is thus

$$I_{KL}(q(\boldsymbol{\theta}|\mathbf{x})||p(\boldsymbol{\theta}|\mathbf{x})) = E_q\left\{\ln\frac{q(\boldsymbol{\theta}|\mathbf{x})}{p(\boldsymbol{\theta}|\mathbf{x})}\right\} = \int q(\boldsymbol{\theta}|\mathbf{x})\ln\frac{q(\boldsymbol{\theta}|\mathbf{x})}{p(\boldsymbol{\theta}|\mathbf{x})}d\boldsymbol{\theta} = \int q(\boldsymbol{\theta}|\mathbf{x})\left[\ln\frac{q(\boldsymbol{\theta}|\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})} + \ln p(\mathbf{x})\right]d\boldsymbol{\theta} = \int q(\boldsymbol{\theta}|\mathbf{x})\ln\frac{q(\boldsymbol{\theta}|\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}d\boldsymbol{\theta} + \ln p(\mathbf{x})\,. \quad (16)$$

The normalising constant $\ln p(\mathbf{x})$ is usually difficult to compute because it requires marginalising the joint density $p(\mathbf{x},\boldsymbol{\theta})$ over $\boldsymbol{\theta}$. The cost function which is actually used is

$$C(\mathbf{x};q) = I_{KL}(q(\boldsymbol{\theta}|\mathbf{x})||p(\boldsymbol{\theta}|\mathbf{x})) - \ln p(\mathbf{x}) = \int q(\boldsymbol{\theta}|\mathbf{x})\ln\frac{q(\boldsymbol{\theta}|\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}d\boldsymbol{\theta}\,. \quad (17)$$

The approximation $q(\boldsymbol{\theta}|\mathbf{x})$ which minimises (17) also minimises (16) because the term $\ln p(\mathbf{x})$ is constant with respect to the approximation $q(\boldsymbol{\theta}|\mathbf{x})$.

In order for ensemble learning to be computationally efficient, the approximation $q(\boldsymbol{\theta}|\mathbf{x})$ should have a simple factorial form. Then the cost function splits into a sum of simple terms which can be computed efficiently.

4.4.2 Bits-back argument

The minimum message length encoding, which was discussed in section 4.3.2, is not decodable, that is, it is not sufficient to actually find x. This is because in order to decode θ, the receiver should know the precision ϵ_θ in addition to the prior knowledge $p(\theta)$, but ϵ_θ was not included in the message. In [131], Wallace and Freeman argued that the values and the accuracies of the parameters are not independent, and one can construct a decodable message with almost the same code length as in minimum message length coding.

Later Hinton and van Camp introduced a "bits-back" argument to show that by a clever coding scheme, a decodable message is obtained without encoding ϵ_θ [44]. The sender uses the following coding scheme:

1. Compute a distribution $q(\theta|x)$ based on the observation x (the algorithm for doing this is defined later).
2. Pick θ from the distribution $q(\theta|x)$ and encode it with a very high prespecified precision (ϵ_θ is close to zero) using $p(\theta)$.
3. Encode x using the encoded θ and $p(x|\theta)$.

It would seem that the sender has had to use a very large number of bits, because the length of the first part

$$L(\theta) = -\int q(\theta|x)\log[p(\theta)\epsilon_\theta]d\theta \quad (18)$$

is very high due to small ϵ_θ. It turns out that this is not the case, however. The trick is that in step 2, the sender can encode a secondary message in the choice of θ. The average length of this message is

$$L(\theta|q) = -\int q(\theta|x)\log[q(\theta|x)\epsilon_\theta]d\theta. \quad (19)$$

Although the length of the first part of the message is very high, most of it is actually used by the secondary message. In the end, the receiver will get those bits back, so to say, and hence the name bits-back argument. The total length of the part which is used for encoding x is

$$L(\theta) + L(x|\theta) - L(\theta|q). \quad (20)$$

It might not be immediately clear that the receiver can decode the secondary message hidden in the choice of θ. In the receiving end, the decoding proceeds as follows:

1. Decode θ from the first part of the message using $p(\theta)$.
2. Decode x from the second part of the message using $p(x|\theta)$.
3. Run the same algorithm as the sender used for computing the distribution $q(\theta|x)$ based on x.
4. Decode the secondary message from θ using $q(\theta|x)$.

Again, the whole point in devising the coding scheme is not the actual transmission of x, but learning about θ. In this case the information about θ comes in the form of distribution $q(\theta|x)$.

Computing all the terms, it turns out that $\log\epsilon_\theta$ appears in both $L(\theta)$ and $L(\theta|q)$, but since these terms appear with opposite signs, ϵ_θ disappears from the equations. The remaining terms are

$$L(\theta) - L(\theta|q) = \int q(\theta|x)\log\frac{q(\theta|x)}{p(\theta)}d\theta. \quad (21)$$

This is the number of bits actually effectively used for encoding θ in the first part of the message. The length of the second part of the message is

$$L(x|\theta) = -\int q(\theta|x)\log[p(x|\theta)\epsilon_x]d\theta. \quad (22)$$

The total length of the message is thus

$$L(\theta) - L(\theta|q) + L(x|\theta) = \int q(\theta|x)\log\frac{q(\theta|x)}{p(x|\theta)p(\theta)}d\theta - \log\epsilon_x\,, \quad (23)$$

which differs from the cost function (17) used in ensemble learning only by the term $\log\epsilon_x$ which can be neglected as it does not depend on the approximation $q(\theta|x)$. The bits-back argument thus results in ensemble learning.

This derivation for ensemble learning is useful because it gives an intuitive meaning for the cost function used in ensemble learning. Different terms of the cost function can be interpreted as the number of bits used for encoding various parameters and the observations. It is then possible to see how many bits have been used for representing different parameters and draw conclusions about the importance of the parameter to the model.

At first glance, it might seem that minimum message length and bits-back coding are quite different, the former using finite accuracy ϵ_θ and the latter a very high accuracy $\epsilon_\theta \approx 0$. It turns out, however, that the minimum message length coding can be seen as a special case of bits-back coding where $q(\theta|x)$ is chosen to be a uniform distribution between

$\theta_0 - \epsilon_\theta/2$ and $\theta_0 + \epsilon_\theta/2$, where ϵ_θ is the finite accuracy used in the minimum message length scheme.

4.5 Specification of the model and priors

So far we have discussed the rules which the learning should ideally follow and some of the practical approximation for these rules. This is roughly the point where the theory ends and practice begins. Bayesian probability theory tells how the beliefs in propositions should be adapted, but it does not indicate what exactly the propositions should be. Neither does it specify the prior beliefs of the learning system. These choices are left for the designer and depend on the problem at hand. This section describes some of the rules of thumb which have been found to be useful guidelines for the design of a learning system (see, e.g., [75, 103, 28]).

4.5.1 Noise models

Due to limitations of computational and storage capacity, it is practically always impossible to take into account all the available knowledge about minor details which possibly have some impact on the observation. It is then reasonable to ignore the details and model only their net effect, which manifests itself in minor fluctuations not predictable from the things included in the model. These fluctuations are called noise.

For real valued parameters, the standard choice of noise model is the Gaussian distribution. The central limit theorem states, roughly, that if a very large number of very small independent fluctuations are summed linearly, then the distribution of the resulting total fluctuation will be Gaussian. In reality, neither the requirement of all the fluctuations being small nor the linearity of summation are perfectly met, but Gaussian noise models are used anyway because they are mathematically very convenient.

4.5.2 Causal structure of the model

Taking into account causal relations of the environment usually results in simpler and computationally efficient models. Take for example a situation where A and B are known to affect C and D, but the effect is causally mediated through E. If E summarises all the knowledge that A and B have about C and D, then C and D are conditionally independent of A and B given E. Mathematically this means that

$$P(CD|AB) = \sum_i P(CD|E_i AB)P(E_i|AB) = \sum_i P(CD|E_i)P(E_i|AB). \qquad (24)$$

It would be possible to consider the situation from the point of view of only the variables A, B, C and D, but then the model would have a dependence from two variables A and B to two variables C and D. Figure 1 represents graphically the introduction of a mediating variable. The nodes correspond to variables and the arrows denote their causal dependences. Such a graph is called a graphical model [73, 59].

In general, a model with more variables but with simpler dependences is computationally more efficient. Introducing mediating variable E simplifies the dependences because either only one variable affects two others, as in $P(CD|E_i)$, or two variables affect one, as in $P(E_i|AB)$. This strategy is a second nature to human beings who constantly try to organise the world by splitting complex dependences into simpler ones by introducing hidden, mediating variables, and therefore it is also usually easy to construct models using

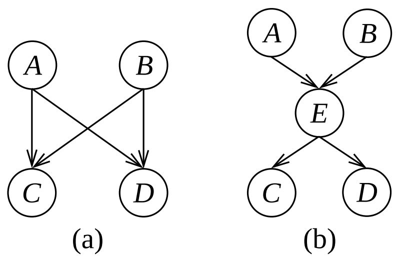

Figure 1: (a) Graphical representation of the causal structure $P(CD|AB)$. (b) Introduction of a mediating variable E simplifies the structure.

the same design principle. The mediating variables are not directly observable but can only be inferred from the dependence structure of the observations. These variables are therefore often called hidden or latent variables [25].

From a computational point of view, the efficiency is caused by the fact that the posterior probability of the unknown variables will be a product of many simple terms. Taking the logarithm will then split the product into a sum of many simple terms. Most methods for approximating the posterior probabilities can make use of this property, including the ML and MAP estimators, the EM algorithm, Laplace's method, ensemble learning and many versions of stochastic sampling.

4.5.3 Supervised vs. unsupervised learning

From a theoretical point of view, supervised and unsupervised learning differ only in the causal structure of the model. In supervised learning, the model defines the effect one set of observations, called inputs, has on another set of observations, called outputs. In other words, the inputs are assumed to be at the beginning and outputs at the end of the causal chain. The models can include mediating variables between the inputs and outputs.

In unsupervised learning, all the observations are assumed to be caused by latent variables, that is, the observations are assumed to be at the end of the causal chain. In practice, models for supervised learning often leave the probability for inputs undefined. This model is not needed as long as the inputs are available, but if some of the input values are missing, it is not possible to infer anything about the outputs. If the inputs are also modelled, then missing inputs cause no problem since they can be considered latent variables as in unsupervised learning.

Figure 2 illustrates the difference in the causal structure of supervised and unsupervised learning. It is also possible to have a mixture of the two, where both input observations and latent variables are assumed to have caused the output observations.

With unsupervised learning it is possible to learn larger and more complex models than with supervised learning. This is because in supervised learning one is trying to find the connection between two sets of observations. The difficulty of the learning task increases exponentially in the number of steps between the two sets and that is why supervised learning cannot, in practice, learn models with deep hierarchies.

In unsupervised learning, the learning can proceed hierarchically from the observations

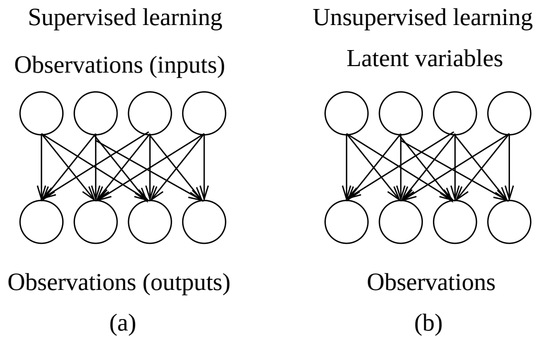

Figure 2: The causal structure of (a) supervised and (b) unsupervised learning. In supervised learning, one set of observations, called inputs, is assumed to be the cause of another set of observations, called outputs, while in unsupervised learning all observations are assumed to be caused by a set of latent variables.

into ever more abstract levels of representation. Each additional hierarchy needs to learn only one step and therefore the learning time increases (approximately) linearly in the number of levels in the model hierarchy.

If the causal relation between the input and output observations is complex — in a sense there is a large causal gap — it is often easier to bridge the gap using unsupervised learning instead of supervised learning. This is depicted in figure 3. Instead of finding the causal pathway from inputs to outputs, one starts building the model upwards from both sets of observations in the hope that in higher levels of abstraction the gap is easier to bridge. Notice also that the input and output observations are in symmetrical positions in the model.

4.5.4 Priors

In order to apply Bayes' rule to updating the beliefs, there have to be some prior beliefs to start with. These prior beliefs are needed for the variables which are at the beginning of the causal chains of the model. In principle the prior probabilities should summarise all the information there is available. In practice the choice of the model structure is often practical and does not reflect the exact beliefs of the modeller and the same holds true for prior probabilities.

Hierarchical model instead of prior. When considering the prior probability for a variable, the first question to ask is whether the variable is really at the beginning of the causal chain. Often there are sets of variables for which there is reason to believe that their values are dependent. This belief is easier to express in terms of model structure than in terms of prior probability. One can postulate a hidden variable which determines the probabilities for the set of dependent variables. The problem of determining a prior is simplified because instead of assigning a separate prior for all variables in the set, only one

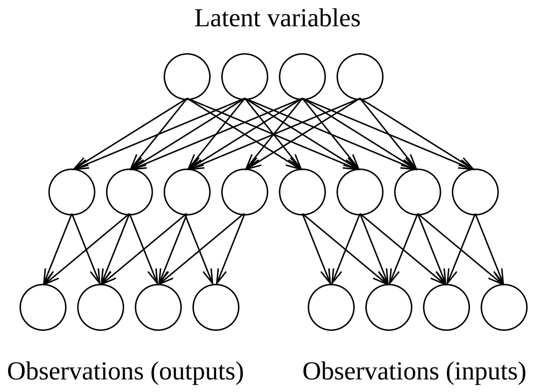

Figure 3: Unsupervised learning can be used for bridging the causal gap between input and output observations. The latent variables in the higher levels of abstraction are the causes for both sets of observations and mediate the dependence between inputs and outputs.

prior is needed for the hidden variable. The process can be iterated and in the end one is usually left with only a few variables which need a prior.

Uninformative priors. Once all the structural prior knowledge is used, there is typically not very much information about the variables at the beginning of causal chains. It is then instructive to consider so called uninformative priors. The name refers to a principle according to which all models should be "given an equal chance" if there is no information to choose between them. The principle has theoretical justification only in cases where symmetries of the problem suggest that the models have the same prior probability. In other cases, the method can be seen as a practical choice which guarantees that the hypothesis space is used efficiently and the learning system is initially prepared to believe in any of the models it can represent.

For real valued parameters, it is usually not a good idea to simply choose a uniform prior for the parameters. The problem again is that probability density has no importance *per se*. A nonlinear transformation of the parameter alters the density differently for different values of the parameter. This means that a uniform density is not uniform after the parameter transformation although the reparameterisation does not alter the model in any way, and shows that it is not possible to assess the uninformative prior for a parameter without knowing what is the role of the parameter in the model.

It is instructive to consider how much the probability distribution which the model gives to the observations changes when the parameters of the model change. Let us take for example the Gaussian distribution parameterised by the mean μ and standard deviation σ. For this parameterisation, the relative effect of change in μ depends on the variance σ: if σ is large, then the change in μ has to be large before the probability assignments for observations change significantly. If σ is small, then a small change in μ causes a relatively large change in the probability assignments. This is depicted in figures 4a and

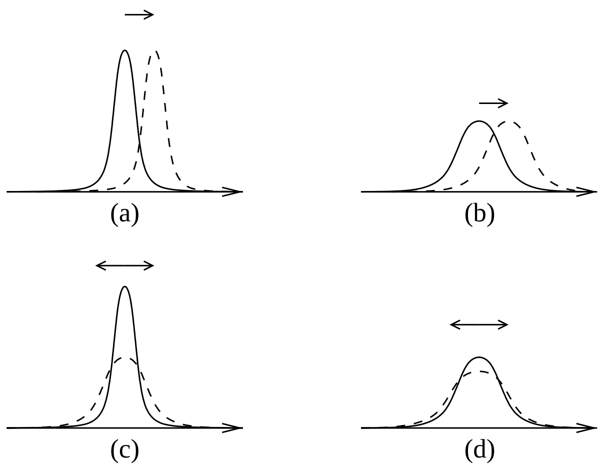

Figure 4: The mean of the distribution changes the same amount in (a) and (b). Similarly, the changes in variance are equal in (c) and (d). The relative changes are greater in (a) and (c).

4b. Similarly, figures 4c and 4d illustrate how the relative effect of change in variance depends on the variance.

The size of change in the observation probabilities caused by the change of parameters can be measured by the Fisher information matrix $\mathbf{I}(\boldsymbol{\theta})$ whose elements $I_{ij}(\theta)$ can be defined as

$$I_{ij}(\boldsymbol{\theta}) = E\left\{ -\frac{\partial^2 \ln p(\mathbf{x}|\boldsymbol{\theta}')}{\partial\theta'_i \partial\theta'_j}\bigg|_{'=} \right\} \tag{25}$$

If the parameters change by $d\boldsymbol{\theta}$, then the Kullback-Leibler distance between the old and new observation probability is $d\boldsymbol{\theta}^T \mathbf{I}(\boldsymbol{\theta}) d\boldsymbol{\theta}/2$. This means that the Fisher information matrix induces a metric in a space of parameters. This is known as the information geometry [1]. In general, it is impossible to find a parameterisation which would make $\mathbf{I}(\boldsymbol{\theta})$ constant because the information geometry is usually not Euclidean.

A uniform density in the information geometry space corresponds to a density which is proportional to $|\mathbf{I}(\boldsymbol{\theta})|^{-1/2}$. This is the uninformative Jeffreys' prior. For the Gaussian density, for instance, this corresponds to a density $p(\mu\sigma) \propto 1/\sigma^2$. This does not produce a proper density because the normalising factor would be infinite. The uninformative prior can nevertheless guide the choice of the prior. For instance, a small adjustment taking into account a finite range for μ and $\ln \sigma$ will result in a prior which can be normalised.

The Fisher information matrix $\mathbf{I}(\boldsymbol{\theta})$ is also important because the posterior densities tend to be approximately Gaussian and have a covariance proportional to $\mathbf{I}(\boldsymbol{\theta})^{-N}$, where N is the number of samples. This property can be utilised for modifying the MAP esti-

mator by multiplying the posterior density by the volume $|\mathbf{I}(\boldsymbol{\theta})|^{-N/2}$. This results in an approximation of posterior probability mass whose maximisation has a more solid theoretical justification than the maximisation of density. It is noteworthy that in models lacking hyperparameters, a combination of Jeffreys' prior and the modified MAP estimate is equal to ML estimation.

In the information geometry space, the parameters tend to have spherically symmetric Gaussian posterior densities and their skewness tends to be smaller than in other parameterisations. This property is useful for parametric approximation of posterior densities because a Gaussian approximation, which is often mathematically convenient, is more valid. The spherical symmetry, on the other hand, can be utilised in gradient descent algorithms because it means that the gradient points to the minimum. The gradient computed in information geometry space is known as the natural gradient and it has been applied to learning neural networks [2].

5 LINEAR FACTOR ANALYSIS AND ITS EXTENSIONS

This section discusses the standard linear factor analysis model and its non-Gaussian, nonlinear and dynamical extensions, as well as the algorithms proposed in the literature for learning these models.

All factor analysis models aim at finding the underlying factors[4] which have generated the observations. Usually the factors are considered to be real valued. The models can be used, for example, for predicting future observations or analysing the nature of the factors or dependences between observations.

The linear model is computationally efficient but can capture only part of the structure in the observations. The extensions aim at capturing more of the structure without overly compromising the computational efficiency.

5.1 Linear Gaussian factor analysis model

According to the model used in ordinary factor analysis, the observations x_i are weighted sums of underlying latent variables. In other words, the dependences between the different components in an observation vector are assumed to be caused by common factors. For consistency with the rest of the thesis, the factors will be denoted by s, although according to the usual convention they would be denoted by f.

The linear summation model is quite simple and it is reasonable to assume there are inaccuracies in the model and many other causes for the observations besides the factors included in the model. The effect of the inaccuracies and other causes is summarised by Gaussian noise n. In anticipation of the dynamic model, the observations are indexed by t referring to time, although in the usual factor analysis model, observations at different time instants are assumed to be independent of each other and the observations therefore need not form a sequence in time.

The linear factor analysis model can be written as

$$x_i(t) = \sum_j A_{ij} s_j(t) + a_i + n_i(t), \tag{26}$$

[4]The factors are also termed hidden causes, latent variables or sources in the literature.

where i indexes different components of the observation vector, j indexes different factors and A_{ij} are the weightings of the factors, also known as factor loadings. The factors s and noise n are assumed to have zero mean. The bias in x is assumed to be caused by a. This is called a *generative model* since it explicitly gives the hypothesis about how the observations were generated.

The model can be written in a vector form as

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{a} + \mathbf{n}(t). \tag{27}$$

Here $\mathbf{x}$, $\mathbf{s}$, $\mathbf{a}$ and $\mathbf{n}$ are vectors and $\mathbf{A}$ is a matrix. This more compact vector form is used by default throughout the thesis.

If the variances of the Gaussian noise terms $n_i(t)$ are denoted by σ_i^2, the probability which the model gives for the observation $x_i(t)$ can be written as

$$p(x_i(t)|\mathbf{s}(t), \mathbf{A}, a_i, \sigma_i^2) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{[x_i(t) - \sum_i A_{ij}s_j(t) - a_i]^2}{2\sigma_i^2}\right). \tag{28}$$

This can also be written simply as

$$\mathbf{x}(t) \sim N(\mathbf{A}\mathbf{s} + \mathbf{a}, \boldsymbol{\sigma}^2), \tag{29}$$

where the vector $\boldsymbol{\sigma}^2$ contains the variances σ_i^2. This notation is used for emphasising that the covariance matrix of $\mathbf{x}(t)$ is diagonal, i.e., the noise on different components is assumed to be independent.

For mathematical convenience, the factors are assumed to have Gaussian distributions in the standard factor analysis model. Recall that the Gaussian distribution emerges if a large number of independent variables are summed linearly. Effectively the Gaussian model for factors then means that the factors are themselves assumed to be caused by various other factors. For many purposes this may be a suitable simplification but it means that the Gaussian factor analysis model is not able to reveal the original independent causes of the observations even if there would be some. Mathematically, this manifests itself in the fact that a multivariate Gaussian distribution with equal variances for all factors is spherically symmetric. Any rotation of the variables will leave the distribution unchanged, and therefore there is a rotational indeterminacy in the model. If the variances of the factors differ, the indeterminacy still exists but the corresponding rotation is non-orthogonal.

From a practical point of view this means that the Gaussian model is able to capture only the second order correlation structure of the components of the observation vectors. Additional criteria can be used to fix the rotation of the matrix $\mathbf{A}$, but it is usually not reasonable to directly interpret the factors as the original independent causes of the observations.

5.1.1 Neural network interpretation of the model

It is somewhat artificial to call the linear factor analysis model a neural network, but it serves as a good starting point for the later development. The structure of neural networks is usually represented graphically by showing the computational elements, neurons, of the network. Each node corresponds to one neuron and the arrows usually denote weighted sums of the values from other neurons. It should be noted that although this representation bears resemblance to graphical models, a graphical model represents the conditional

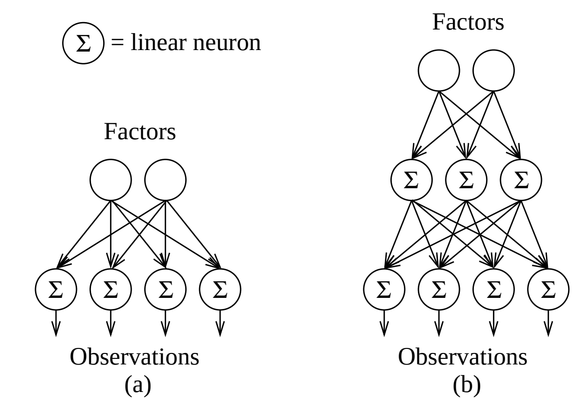

Figure 5: (a) The linear factor analysis model can be interpreted as a neural network with one layer of linear neurons. The arrows represent connections with weights A_{ij} attached to them. (b) A multi-layer structure can be replaced by a single layer of linear neurons.

dependences while the neural network representation shows the computational structure. In general, these representations are therefore different.

The linear factor analysis model can be represented as a neural network with two layers[5]. On the first layer there are the factors and the second layer consists of linear neurons which compute a weighted sum of their inputs. A network interpretation of a model with two-dimensional factors and four-dimensional observations is depicted schematically in figure 5a. The weights A_{ij} are shown as links between the nodes but the biases a_i are not shown.

Linear neurons as building blocks for larger networks are too simplistic because adding extra layers of linear neurons does not increase the representational power of the network. This is easily seen by considering the model

$$\mathbf{x}(t) = \mathbf{B}(\mathbf{A}\mathbf{s}(t) + \mathbf{a}) + \mathbf{b} + \mathbf{n}(t), \tag{30}$$

shown in figure 5b. By setting $\mathbf{A}' = \mathbf{B}\mathbf{A}$ and $\mathbf{a}' = \mathbf{B}\mathbf{a} + \mathbf{b}$ the model can be written as

$$\mathbf{x}(t) = \mathbf{A}'\mathbf{s}(t) + \mathbf{a}' + \mathbf{n}(t), \tag{31}$$

which, interpreted as a neural network, has only one layer of linear neurons.

[5]There are two conventions for counting the layers. Here the factors are counted as one layer although they are not computational elements. The alternative is to count only layers of computational elements. Both conventions agree on how to count hidden layers.

5.1.2 Algorithms

Factor analysis is widely used in the social sciences and there is an extensive literature describing various methods for estimating the factors and then rotating them in order to yield solutions which have simple interpretations [122, 110, 35, 25, 5]. Principal component analysis (PCA) is often used as the first stage for finding the factors [63, 58]. We shall call PCA and related methods signal transformation (ST) approaches. These methods define a criterion on the factors and then find a transformation from observations to factors which optimises the criterion. In PCA, the criterion is that the resulting factors are uncorrelated and have maximal variance. A further requirement of orthogonal $\mathbf{A}$ is required to make the problem well determined.

In many cases ST approaches yield a fair approximation to the result which would be gained by the Bayesian approach to estimating the factors. Almost always, however, the ST methods provide only point estimates. From the point of view of this thesis, reference [9] is relevant because it uses ensemble learning for estimating the posterior probabilities of the factors and the mapping $\mathbf{A}$.

There are several criteria for the rotation. Varimax, one of the most popular rotation criteria, aims at maximising the sparsity of the matrix $\mathbf{A}$. In general, the goal of the rotation is to find a meaningful interpretation of the resulting factors.

5.2 Non-Gaussian factors

The Gaussian model for the factors leads to inability to separate the underlying causes from each other. This is remedied by considering models where the factors are not restricted to be Gaussian but have a more general model. These models can be called independent factor analysis (IFA) models or independent component analysis (ICA) models depending whether they are seen as extensions of factor analysis or principal component analysis models (for textbook accounts see, e.g., [76, 34]).

The independence refers to the fact that the non-Gaussian model of the factors enables factor analysis to find the underlying statistically independent factors which have generated the observations linearly if they exist. In the ICA community, the factors have been traditionally called sources because they are not just some arbitrary combinations of the underlying causes, but ideally at least, the original independent causes. The practical techniques for estimating the sources are known also as blind source or signal separation.

Other properties apart from fixing the rotation of the factors are largely the same for the linear non-Gaussian factor analysis as for the linear Gaussian factor analysis. A linear multi-layer model, for instance, can still be brought down into one linear layer.

5.2.1 Algorithms

Several algorithms for independent component analysis have been proposed in the literature [61, 17, 95, 51, 3, 52]. Many of the algorithms use the signal transformation approach with the criterion that the resulting sources be as statistically independent as possible. These algorithms can be seen as extensions of PCA. Simplifying assumptions can provide very efficient algorithms, such as FastICA [55, 51]. Some of the algorithms use the natural gradient which was discussed in section 4.5.4.

MAP estimation does not work for the IFA model unless the linear mapping $\mathbf{A}$ is suitably restricted. This is because the width of the peak of the posterior probability density of the factors, and thus the posterior probability mass, depends on the matrix $\mathbf{A}$.

However, it is possible to take into account the width of the posterior. If the variance of noise $\mathbf{n}(t)$ is assumed to be the same for all observations, then the posterior volume is proportional to the posterior density and inversely proportional to the determinant $|\mathbf{A}|$. This has been used in [99], although the method is given a different interpretation.

In most algorithms, a point estimate is used for the linear mapping $\mathbf{A}$. Hyvärinen *et al.* have shown that many of these algorithms suffer from overfitting [56]. Publication II describes how ensemble learning can be applied to IFA. Point estimates are not used for any parameters and therefore the algorithm is not prone to overfitting. In publication VII, a new interpretation of the FastICA algorithm is given, which allows one to use the same idea with ensemble learning, thus yielding a Bayesian version of the FastICA algorithm. The treatment of the posterior factor distributions is improved from Publication II by utilising the method applied in [3].

5.3 Nonlinear mapping

An obvious way to extend the representational power of the linear factor analysis model is to consider the case where the factors can have a nonlinear effect on the observations

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t). \tag{32}$$

Here we replaced the affine mapping $\mathbf{As} + \mathbf{a}$ by an arbitrary function $\mathbf{f}(\mathbf{s})$.

The nonlinear mapping does not change the considerations about the density of the factors. With the Gaussian model for the factors, the nonlinear mapping yields an extension of the ordinary factor analysis, that is, there is the same indeterminacy relative to the rotation of the factors. Nonlinear mapping with non-Gaussian model for the factors yields an extension of independent factor analysis which is able to recover the underlying factors if they exist.

5.3.1 Multi-layer perceptron networks

The nonlinear model is very general and can, in principle, represent almost anything. Again there is the trade-off between representational power and efficiency, raising the practical problem of finding a suitably restricted subset of functions $\mathbf{f}$ which would have good representational capacity but would also form a space with a structure regular enough to enable efficient learning in practice.

The choice of the set of functions $\mathbf{f}$ depends on the problem at hand, but a possible choice is the multi-layer perceptron (MLP) network [111, 8, 39]. It has often been found to provide compact representations of mappings in real-world problems. The MLP network is composed of neurons which are very close to the ones represented in the case of the linear network. The linear neurons are modified so that a slight nonlinearity is added after the linear summation. The output c of each neuron is thus

$$c = \phi\left(\sum_i w_i a_i + b\right), \tag{33}$$

where a_i are the inputs of the neuron and w_i are the weights of the neuron. The nonlinear function ϕ is called the activation function as it determines the activation level of the neuron. This refers to interpreting the activation as the pulse rate of biological neurons.

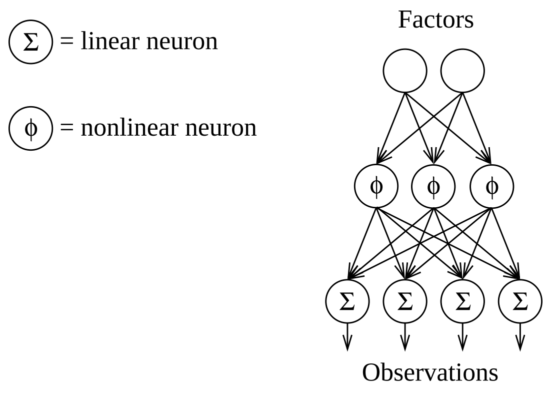

Figure 6: A graphical representation of the computational structure of an MLP network with one hidden layer of nonlinear neurons.

Due to the nonlinear activation function, a multi-layer network is not equivalent to any one-layer structure with the same activation function. In fact, it has been shown that one layer of suitable nonlinear neurons followed by a linear layer can approximate any nonlinear function with arbitrary accuracy, given enough nonlinear neurons [49]. This means that an MLP network is a universal function approximator.

The activation functions most widely used are the hyperbolic tangent $\tanh(x)$ and logistic sigmoid $1/(1 + \exp(-x))$. They are actually related as $(\tanh(x) + 1)/2 = 1/(1 + \exp(-2x))$. These activation functions are used for their convenient mathematical properties and because they have a roughly linear behaviour around origin, which means that it is easy to represent close-to-linear mappings with the MLP network.

Figure 6 depicts an MLP network with one layer of linear output neurons and one layer of nonlinear neurons between the input and output neurons. The middle layers are usually called hidden layers. Notice that a graphical model of the conditional dependences would not include the middle layer because the computational units are not unknown variables of the model whereas the weights would be included as nodes of the model.

The mapping of the network can be compactly described by (34).

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t) = \mathbf{B}\boldsymbol{\phi}(\mathbf{A}\mathbf{s}(t) + \mathbf{a}) + \mathbf{b} + \mathbf{n}(t). \tag{34}$$

According to the usual notation with MLP networks, the vector $\boldsymbol{\phi}$ denotes a vector of functions which each operate on one of the components of the argument vector.

5.3.2 Algorithms

The MLP networks belong to the standard tool box of modern neural networks research. Almost always, however, they are used for supervised learning, to the extent that often MLP networks are thought to be suitable only for supervised learning.

Since the signal transformation approach has been successfully applied to the derivation of efficient, practical algorithms for linear PCA and ICA, it is natural to try the same approach for nonlinear models as well. Signal transformation approaches have been proposed for nonlinear independent component analysis using MLP networks in [13, 86, 134, 125, 135, 126, 87, 45]. The flexibility of the MLP network makes overfitting a serious problem when point estimates are used. In order to alleviate overfitting, many of the ST approaches need to restrict the structure of the MLP network.

ML estimation has been applied to a nonlinear factor analysis model containing an MLP network in [94]. The use of a point estimate causes the same overfitting problem as with ST approaches. In [85], stochastic sampling has been used for learning a nonlinear factor analysis model using an MLP network. Due to the large number of unknown variables in the model, learning is extremely slow. The dynamic nonlinear models proposed in [31, 12] are discussed in section 5.4. Section 6 summarises the nonlinear factor analysis algorithm developed in this thesis.

Self-organising map (SOM) [65] and the related method generative topographic mapping (GTM) [10] can be interpreted as nonlinear factor analysis models [42, 98, 79, 100]. The parameterisation used for the mapping is rather different from that used in linear models and MLP networks. Instead of defining weighted products, each neuron specifies a point in the observation space. The neurons are arranged in a low-dimensional grid and the grid can be interpreted as the latent space, that is, the coordinates of a neuron specify the values of the factors.

Self-organising maps have been found useful, computationally efficient tools for visualising the structure of data sets because they find a two-dimensional representation for high-dimensional observations. However, the parameterisation which is based on specifying points in the observation space is not well suited for factor analysis when the dimension of the latent space is even moderately high. This is because the number of neurons in the grid grows exponentially as a function of the dimension of the latent space. For linear models and MLP networks the number of parameters grows linearly as a function of the dimension of the latent space and they are therefore better suited for learning models with a large number of latent variables.

Auto-associative models. The signal transformation approach aims at estimating the recognition mapping from observations to factors while generative learning models the mapping from factors to observations. The third alternative, auto-associative learning, estimates both at once. The basic idea is to find a mapping, defined by an MLP network for instance, from observations to themselves through an information bottleneck. This forces the network to find a compact coding for the observations.

Learning is supervised in the sense that both the inputs and the outputs of an MLP network are specified. This also means that learning is exponentially slow in the number of hidden layers. It is, however, possible to use the same strategy as in unsupervised learning; gradually tightening bottlenecks can be added in the middle of the network as the learning proceeds. In any case, the learning is slower than with generative models because both the recognition mapping and the generative mapping need to be learned separately. In generative learning the recognition mapping can be computed from the generative mapping by Bayes' rule.

The information bottleneck is usually implemented simply by restricting the number of hidden neurons in the middle layer of the network. However, this alone does not restrict the information content of a real number because the amount of information depends on the accuracy of coding. Using the minimum message length inference or bits-back argument which leads to ensemble learning, it would be possible to accurately measure the information content but the resulting algorithm would no longer profit anything from restricting the recognition mapping to be the one implemented by the MLP network. The experiments using an MLP network for recognition mapping in [128] show that gradient descent based inversion of the generative mapping for new observations can profit from the initial guess provided by an MLP network estimating the recognition mapping, but the performance of an MLP network is poorer than that found when gradient descent alone is used.

Examples of the use of auto-associative MLP networks can be found in [40, 48, 47]. From the point of view of this thesis, references [48, 47] are particularly relevant because they use flat minimum seach [46], a method which bears resemblance to minimum message length inference, for measuring the complexity of the MLP network. However, the complexity of the factors is not measured.

5.4 Dynamic factors

In many cases, observations form a sequence in time, and it is useful to extend the factor analysis model to take into account the dynamical behaviour of the factors. In physics and signal processing, these models are in wide use and are called state space models, that is, the factors in this context are called states.

In the general nonlinear form, the model describes the sequence of observations $\mathbf{x}(t)$ which have been caused by a sequence of states $\mathbf{s}(t)$ through a mapping $\boldsymbol{f}$. In physics, the state dynamics is often presented as partial differential equations. For our purposes, discrete-time difference-equations are more appropriate. According to the model, the state vector is assumed to be mapped nonlinearly on the consecutive state vector through the function $\mathbf{g}$. Since the model does not aim at representing the complete physical state of the universe, the state is certainly affected by some other factors besides the previous state. These external influences are summarised in a noise model $\mathbf{m}(t)$, which is also called the innovation process. The dynamic mapping thus has a form very similar to the mapping from states to observations:

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t) \tag{35}$$

$$\mathbf{s}(t+1) = \mathbf{g}(\mathbf{s}(t)) + \mathbf{m}(t). \tag{36}$$

The distribution of the innovation process $\mathbf{m}(t)$ determines whether the model resembles ordinary or independent factor analysis; the distribution of $\mathbf{m}(t)$ has to be non-Gaussian in order for the model to be able to reveal the possible underlying independent innovation processes. The mappings are also often assumed to be functions of observed exogenous inputs $\mathbf{u}(t)$: $\mathbf{f}(\mathbf{s}(t), \mathbf{u}(t))$ and $\mathbf{g}(\mathbf{s}(t), \mathbf{u}(t))$. This corresponds to having a mixture of supervised and unsupervised learning.

5.4.1 Algorithms

A substantial amount of research has been conducted on estimating the states $\mathbf{s}(t)$ under the assumption that the mappings $\mathbf{f}$ and $\mathbf{g}$ are known. This is called Kalman filtering if the mappings are linear and extended Kalman filtering if the mappings are nonlinear. Several textbooks give introductions to the field, for instance, [38, 121, 88].

Many of the algorithms for unsupervised learning of the generative mapping $\mathbf{f}$ can be extended to learn also the state dynamics $\mathbf{g}$. An obvious way is to first learn a static model with only $\mathbf{f}$ and then use supervised learning to learn $\mathbf{g}$. However, this procedure has the disadvantage that during the learning of $\mathbf{f}$, dynamic structure is not taken into account and therefore the resulting representation does not necessarily make efficient use of the dynamics.

An early application to learning both $\mathbf{f}$ and $\mathbf{g}$ together can be found in [118]. However, both mappings are assumed to be linear. A linear generative mapping $\mathbf{f}$ and nonlinear dynamic mapping $\mathbf{g}$ have been used in [16]. Some training samples where the states are observed are assumed to be available and therefore the method is not wholly unsupervised.

Fully nonlinear learning algorithms have independently been proposed in [31, 12]. In [31], Gaussian radial basis functions (RBF) [90] are used for modelling the mappings $\mathbf{f}$ and $\mathbf{g}$ and the EM algorithm [21] is used for learning parameters of the linear mapping in the RBF model, in other words, the nonlinearities are not adapted. The structure of the Gaussian RBF model allows analytical computation of the expectations required for adapting the linear mapping in the model which makes the approach interesting. The drawback is that since the nonlinearities are not adapted, the required number of hidden neurons is exponential in the dimension of latent space.

MLP networks have been used in [12] to model the mappings $\mathbf{f}$ and $\mathbf{g}$. First the posterior density of the states is approximated and then samples are taken from the posterior and used to estimate the parameters of the MLP network with the ordinary backpropagation algorithm [111] (see also, e.g., [39, 8]). Point estimates for the parameters of the mappings and Gaussian models for the innovation process were used in both [31, 12].

Publication VIII shows how the nonlinear factor analysis algorithm developed in this thesis can be extended to take into account the dynamics of the factors.

6 BAYESIAN NONLINEAR FACTOR ANALYSIS

The motivation for developing methods tailored for unsupervised learning is that to learn large hierarchical models, unsupervised learning is the most promising approach. A characteristic property of unsupervised learning is the potentially large amount of unknown variables. This is because the latent variables need to be estimated for each observation separately. In Bayesian learning, the posterior distribution of these unknown variables requires to be estimated.

In this section, a nonlinear extension of the factor analysis model is used as an example of how ensemble learning can be applied to unsupervised learning of this kind of model and why it should be used in the first place. Ensemble learning is discussed at length in publication IV and a detailed description of its application to nonlinear factor analysis by MLP network is given in publication V. In this section, only the main points are summarised.

6.1 Model

The nonlinear factor analysis model described in publication V has the following structure:

- The mapping from factors to observations is modelled by an MLP network with one hidden layer as described by (34).

- The factors can have Gaussian or mixtures-of-Gaussians models. This corresponds to a nonlinear extension of linear factor analysis or linear independent factor analysis, respectively. The factors are assumed to be independent.

- Hierarchical models are used for describing the prior information about the parameters. Gaussian distributions are used extensively.

- The variance parameters of the Gaussians are parameterised by the logarithm of the standard deviation. This yields a roughly Gaussian posterior probability for these parameters.

6.2 Why simple methods fail

The standard backpropagation algorithm has been used successfully for estimating the parameters of an MLP network for a long time (see, *e.g.*, [39, 8]). It therefore requires some explanation why it cannot be used in this case. The basic reason is that in supervised learning, only the weights of the MLP network are unknown while in unsupervised learning the inputs to the MLP network are also unknown.

Posterior probability mass is proportional to the volume of the posterior peak which in turn is proportional to the posterior uncertainty of the unknown variables. With simple linear models used in factor analysis and independent factor analysis, it is possible to constrain the linear mapping $\mathbf{A}$ so that the posterior uncertainty of the factors is roughly constant or bounded from below. Even then the simple linear independent factor analysis algorithms suffer from overfitting as shown in [56]. With nonlinear models it is far more difficult to ensure that the posterior uncertainty of the factors is constant without posing very restricting conditions on the allowed nonlinearities (however, see [20] for an example) and the models are, in any case, more vulnerable to overfitting because they have more parameters to be estimated.

The severity of overfitting is roughly proportional to the number of those unknown variables in the model which are given point estimates and inversely proportional to the number of observations. In supervised learning, a sufficiently large number of observations can reduce overfitting, assuming that the model structure is predetermined. In unsupervised learning, increasing the number of observations cannot decrease the ratio below the dimension of latent space divided by the dimension of observations. This is because for each observation $\mathbf{x}(t)$, the corresponding values of latent variables $\mathbf{s}(t)$ need to be estimated separately.

The EM algorithm can be used for learning nonlinear latent variable models as shown in [31]. The number of variables which are assigned point estimates is then comparable to supervised learning. Most of the computational cost corresponds to the computation of the distribution of the factors and the extra computational cost of assigning posterior distributions to the rest of the parameters as well is not very high.

6.3 Approximation of the posterior

The result of learning is an approximation of the posterior probability of all the unknown variables given the observations. The unknown variables are the factors $\mathbf{s}(t)$, the parameters of the mapping $\mathbf{f}$, variance parameters for factors and observation noise and the parameters of the hierarchical prior. The approximation is needed because the posterior joint probability of all the unknown variables is very complex due to the large number of unknowns and the complex structure of the model.

In most publications of this thesis, the approximation of the posterior is assumed to have a maximally factorial form, that is, all the unknown variables are assumed to be independent given the observations. This can be seen as a necessary and sufficient extension to point estimates which is sensitive to posterior probability mass instead of probability density. Publication VIII shows how some of the most important posterior correlations of the variables can be included in the approximation without compromising the computational efficiency. Notice that although the variables are assumed to be independent *a priori*, they are dependent *a posteriori* because the observations induce dependences between them.

Publication I presents the methods in minimum message length framework and therefore uses a uniform distribution as the approximation for the posterior. Other publications use the Gaussian distribution which is in general a better approximation to posterior densities. It is also often possible to choose a parameterisation, such as the logarithmic parameterisation of the variance or the "softmax" parametrisation of the mixture coefficients [84], which makes the Gaussian approximation even more valid.

The cost function in ensemble learning is the Kullback-Leibler information between the posterior probability and its approximation. Due to the simple factorial form of the approximation, the cost function and its derivatives can be computed efficiently. The required computations resemble very much the ones which would be carried out using the standard backpropagation algorithm for estimating the unknown variables of the model. The most notable difference is that scalar values are replaced by probability distributions of the values.

During learning, the approximation of the posterior is adapted by a modification of gradient descent which utilises the structure of the problem as explained in publication V. The difference to ordinary point estimation is that the weights and factors are characterised by their mean and variance. This is important as then the algorithm is sensitive to the probability mass in the posterior probability instead of being sensitive to probability density. Figure 7 illustrates the adaptation of the posterior probability density instead of a point estimate for an unknown variable of the model.

6.4 Automatic pruning

A convenient by-product of using a factorial approximation of the posterior density in ensemble learning is that unused parts of the model are effectively pruned away. The reason for this is that the learning process aims at fitting the approximation to the true posterior. It is usually the case that the model has some indeterminacies, which basically means that several different values for the variables in the model yield exactly the same probability for the observations. It is then impossible to determine the variables based on observations.

For ensemble learning this can be a benefit because it allows a choice of parameter values which make the factorial assumption of the posterior density more valid. In other words, extra degrees of freedom in the model can be used for improving the approximation of the posterior density.

If the model has more parameters or factors than are required, some of them are not well determined which will be reflected in having equal posterior and prior distributions. In a general case where the variables have posterior correlations, there are some directions in the variable space which are well determined and others which are not. Figure 8a gives an example of such a situation. The difference of the two parameters is well determined while the value of their sum is uncertain.

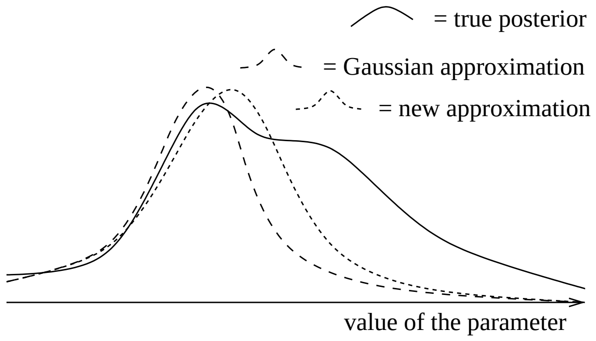

Figure 7: During adaptation, the approximation of the posterior is fitted to the true posterior distribution. The dashed line shows schematically how the approximation could change in one iteration step. The mean and variance of the approximation are adapted to fit the true posterior better.

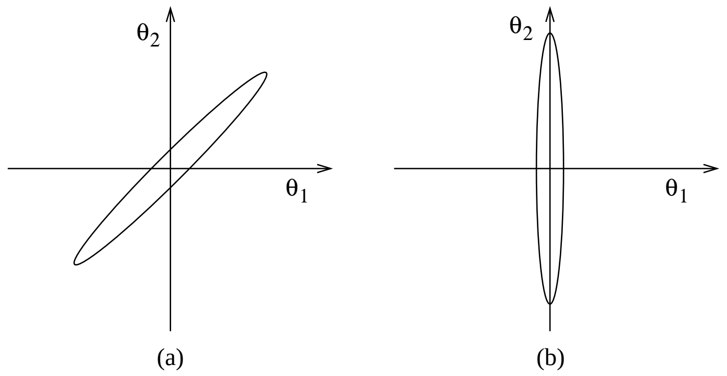

Figure 8: Schematic illustrations of posterior densities of two parameters θ_1 and θ_2. (a) The sum of the parameters is not determined while the difference is. (b) After rotation, θ_1 is left determined and the undetermined θ_2 can be pruned away.

If indeterminacies have produced degrees of freedom which allow rotation of the parameter space, then ensemble learning will try to make the variables posteriorly independent as shown in figure 8b. This means that the variables tend to be either well determined or not determined at all. Equation (21) which is the Kullback-Leibler information between the prior and approximated posterior density of a variable can be used for assessing whether the variable is actually used by the model. If the posterior is close to the prior, the variable is not well determined. From the coding point of view, equation (21) can also be interpreted as the number of bits used by the model to represent that variable. If very few bits are used to represent a variable, it is not needed by the model and can be pruned away without affecting the model.

6.5 Results

Most of the experimental results on nonlinear factor analysis are summarised in publication V[6]. Reference [128] presents many of the same experiments and in addition an experiment which shows how factors can be estimated for new observations which have not been present during learning. Publication VIII reports experiments with the dynamic extension of the nonlinear factor analysis algorithm.

In general, the experiments have verified that ensemble learning can be successfully applied to nonlinear factor analysis using MLP networks. Ensemble learning avoids the problems related to overfitting which is a severe problem for simpler algorithms. It is also easy to optimise the structure of the model simply by minimising the cost function. The number of factors and hidden neurons of the MLP network, for instance, can be reliably optimised.

It is well known that MLP networks have local minima (see, e.g., [39, 8]). This seems to be a nearly unavoidable consequence of using complex models with rich representational capacities. It is recommendable to try several different random initialisations of the network and choose the result which minimised the cost function.

The experiments with artificial data show that if the structure of the observations matches the model, then the algorithms developed in this thesis are able to reveal the original, independent factors. In many realistic cases there is reason to believe that the underlying structure of the observations is more accurately described as a nonlinear than linear mapping from underlying factors to observations. Experiments with measurements from an industrial pulp process reported in publication V have verified this at least for that data set. With a nonlinear model, far fewer factors are needed to represent the data than with a linear model.

7 DISCUSSION

7.1 Biological relevance

The methods developed in this thesis can be used for data analysis and are therefore useful as such. However, the research which has led to these methods has also been inspired by the biological brain. This section discusses some of the connections between the brain and the nonlinear factor analysis algorithm.

[6]Matlab code for running the simulations is available at `http://www.cis.hut.fi/projects/ica/bayes/`.

Although the biological brain does not implement any specific mathematically exact algorithm, Bayesian learning or others, the framework of Bayesian probability theory is appropriate for interpreting many of the different functions of the brain (see, e.g., [64]). Visual perception of shape from shading, for instance, was successfully analysed from the Bayesian point of view in [27].

On one hand, findings from the neurosciences can teach us how to build better algorithms for learning models from observations, and on the other hand, development of these algorithms can give valuable insight for interpreting the relevance of the findings. The model space and priors used by the brain, as well as the particular computational approximation can be interpreted to be implicitly defined by the genes because they give the brain the instructions on how to react to different environmental stimuli. The hypothesis space implicitly defined by the brain is evidently huge, computationally efficient and allows a rich representation of the environment.

It is obvious that learning is not the only function for the biological brain and explaining the structures found in the brain requires account to be taken of prediction and action. It seems plausible that the cerebral cortex is involved in both learning and inference or planning because the cortex is activated during imaging tasks [68]. However, the following concentrates only on learning. For textbook accounts on the brain see, e.g., [69, 104, 62].

7.1.1 Cerebral cortex and generative models

The cerebral cortex maintains a model of the environment. Its perceptual machinery constantly seeks explanations for sensory inputs [4]. It seems likely that unsupervised learning with generative models is the appropriate interpretation which helps understanding many aspects of the learning taking place in the cortex. It is known that the abstraction level of representation in the cortex gradually increases from primary sensory areas to higher cortical areas (see, e.g., [127]) where it is possible to find neurons that react as if they would bridge different modalities in the manner depicted in figure 3.

At least two independent experimental findings and one theoretical argument supports generative learning over signal transformation or auto-associative learning. First, the signal transformation assumption does not predict backward connections and auto-associative learning predicts roughly equal amounts of forward and backward connections between different levels of brain areas (forward connection here means the direction from sensory to higher areas). In reality, there are up to ten times as many backward connections between cortical areas as there are forward connections (see, e.g., [26]). This fits well with the generative learning assumption, because backward connections define the meaning of the representation and forward connections carry the gradient information or error signals needed to update the representation and for that purpose, gradient information needs not be very accurate.

Second, the temporal behaviour of the forward signals from visual area V1 to V2 has been shown to fit well to the interpretation of error signals [105], and also to be modulated appropriately if the activity of V2 is blocked [50].

Third, in generative learning, the number of connections needs to be proportional to the number of neurons N while in the signal transformation approach the number of connections requires to be proportional to N^2 which is much more than actually observed in the brain. In order for a large network of neurons to learn, the neurons need to have a way of informing each other when something is already learned to avoid all neurons from learning the same thing. In generative learning, the forward signals carry the error signals and therefore a persistent signal indicates that no other neuron is representing the input.

In the signal transformation approach, each neuron has to inform all other neurons that they have learned something. There are short range inhibitory lateral connections in the cortex, but not enough to support signal transformation interpretation [32]. The purpose of the inhibitory lateral connections can be, for instance, to assist in finding the correct activations when error signals arrive [74, 29]. Only those neurons need to be laterally connected which represent very similar things (as defined by the backward connections) thus explaining the short range of the lateral connections.

7.1.2 Cortex and nonlinear factor analysis

Interpreted pedantically, MLP networks used in the nonlinear factor analysis model in this thesis are not realistic neuronal models of the brain. It is clear that the neurons in the brain are quite different from the ones used in a typical MLP network, but it is possible to see many of the aspects of the MLP networks as abstractions of the biological brain. First, biological neurons fire action potentials and the output activation of the neurons in an MLP network is an idealisation of the firing rate. Secondly, there are various kinds of interneurons in the cortex which the MLP networks lack. However, it is possible to view each neuron of an MLP network as an abstraction of one microcolumn or one pyramidal neuron and several interneurons. In [105], the microcolumns have been interpreted as Kalman filters.

The representational capacity of the factor analysis model is weaker than that of the biological brain. To be more specific, the model cannot adequately represent objects and therefore cannot represent relations between objects. There is much evidence suggesting that the temporal structure of the firing of biological neurons in the cortex carries information related to object representations. In the brain, the object representations seem to be linked with synchronous activity of the neurons; the neurons which represent features belonging to one object fire synchronously [24, 37, 36].

Representing only the firing rate of the neurons misses the temporal information and results in an inability to represent the binding between features which would define objects. Consequently, the representations also lack relations between objects. This is one of the most serious drawbacks of most of the current neural network models. Many attempts have been made to build models with neurons whose activity is represented by firing of pulses as with biological neurons [80]. Since the real valued abstraction of the firing rate has served as a good computational simplification, modifications which represent both the activity and the phase or binding of the neurons have also been proposed (e.g., [23]).

Whether the standard real valued neurons are replaced by firing or more abstracted neurons, it can be hoped that the lessons learned from the simple neurons used in most neural network models today will still be useful. This seems realistic since so many aspects of the real brain can be interpreted from the point of view of simple real valued neurons. If the brain is the kind of virtual machine built on parallel networks which Dennet proposes [22], then it seems possible that we can also design similar machines by starting with the parallel networks discussed in this thesis and then modify them slightly so that sequential processing is implemented.

7.1.3 Structural development

It takes years for the human brain to mature. Although neural networks researchers are usually not that patient with their own models, the process by which the connections in the brain grow and adapt can give useful hints on how large hierarchical artificial neural networks can be learned.

Two basic principles have shown their usefulness in unsupervised learning of MLP networks. The first is that it is easier to start with a large network and then prune away unused parts. This is similar to what happens in the biological brain where a significant number of the neurons die during early development (see, e.g., [69, 104, 62]). There is evidence that the neurons which die are the ones which fail to find something reasonable to represent. The same behaviour is automatically implemented by ensemble learning as discussed, for example, in publication III.

The second principle is the critical period of development during which the connections are established. This period starts earlier in the lower areas close to sensory areas and then proceeds to higher levels [104]. Experience with ensemble learning for unsupervised MLP networks showed that an algorithm which is capable of pruning connections and neurons needs this procedure for learning hierarchical representations. If the critical period would be too early, the neurons would be pruned before they have a chance to learn to represent anything useful because their lower areas would not yet have established a sensible representation.

Although the neurons in the cortex can have 1,000–10,000 connections [69], one neuron connects only to a very small fractions of all neurons, that is, the connectivity is sparse. The MLP networks studied in this thesis had only tens of neurons and it was therefore feasible to use fully connected layers. However, ensemble learning can also accommodate the pruning of connections in large networks. The dynamic model of the factors in publication VIII exhibited signs of sparse connectivity.

In ensemble learning, the pruning is caused by the pressure to make the posterior probability of the parameters and factors of the model as independent as possible. It can be argued that this is also a useful strategy for the brain as it would be difficult to keep track of the posterior dependences of the activities of all the neurons in the brain and even more difficult to model the posterior dependences of the strengths of the synapses.

7.2 Future trends

A great deal of work still requires to be undertaken in the development of the methods discussed in this thesis. There are many extensions of the basic MLP structure which can be utilised. For instance, bilinear neurons that compute weighted products of two inputs could be useful. In some cases, there is a strong reason to believe that the mapping from factors to observations includes this type of functions, but with neurons having simple weighted sums it is difficult, albeit not impossible, to represent such mappings.

One of the strong practical advantages of using the Bayesian framework for learning is that it is easy to combine different models and algorithms. An example of this was seen in the development of the treatment of a non-Gaussian factor distribution, where the method used in publication II was replaced by the one borrowed from [3]. One aspect which could clearly be improved is taking into account the posterior dependences of the factors. The simpler approximation could still be used for providing a good initial guess.

This thesis concentrates on real valued representations, but the extensive research conducted with discrete valued representations and observations can be utilised because within the Bayesian framework, it is straight-forward to combine different models. Likely candidate models include belief networks [101], sigmoid belief networks [91, 113], hidden Markov-models [83], switching state-space models [30] and mixture models [89].

Missing observations pose no problem in the Bayesian framework as they can be treated like any other unknown parameter of the model. This enables unsupervised learning to be used for similar tasks as supervised learning but without the requirement to prespecify which of the observations are inputs and which are outputs.

In many large problems the prior knowledge at hand suggests a modular structure for the MLP network which can be taken into account. It should also be easy to develop automatic procedures for pruning and model selection because the cost function in ensemble learning can be reliably used for model selection. When learning large models, this should be useful, as well as in learning procedures where layers of neurons are added to the network one by one.

Factors governing the variance of other factors seem likely candidates for building-blocks for mappings whose learning is computationally efficient but which are representationally powerful. These models are inspired by the properties of complex cells found in the visual area V1 (see, e.g., [62]), whose behaviour appears to match well with this function. Models which have factors resembling complex cells have been proposed, for example, in [66, 14, 29, 54, 53].

In order to match human capabilities, the models will need to represent objects and relations between objects. Not enough is known about the representation of these things in the biological brain in order to utilise the knowledge directly for artificial neural network models. The work done in traditional artificial intelligence research (see, e.g., [112]) can, however, give good starting points.

One of the most important application areas for the methods developed in this thesis and a fruitful source of new ideas will probably be the problem of adaptive process control because many processes have natural representations in terms of real valued state-spaces and the controlled signals are also often analogue. Learning models of the environment based on observations is also one of the most demanding problems in reinforcement learning where an autonomous agent is trying to make decisions based on external rewards (see, e.g., [124]).

REFERENCES

[1] S. Amari, *Differential-Geometrical Methods in Statistics.* Springer-Verlag, 2nd ed., 1990.

[2] S. Amari, “Natural gradient works efficiently in learning,” *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

[3] H. Attias, “Independent factor analysis,” *Neural Computation*, vol. 11, no. 4, pp. 803–851, 1999.

[4] H. B. Barlow, “Cerebral cortex as model builder,” in *Models of the visual cortex* (D. Rose and V. G. Dobson, eds.), pp. 37–46, John Wiley & Sons, 1985.

[5] A. Basilevsky, *Statistical Factor Analysis and Related Methods: Theory and Applications.* John Wiley & Sons, 1994.

[6] R. A. Baxter and J. J. Oliver, “MDL and MML: Similarities and differences,” Tech. Rep. TR 207, Department of Computer Science, Monash University, Australia, 1994.

[7] J. M. Bernardo and A. F. M. Smith, *Bayesian Theory.* Wiley, 1994.

[8] C. M. Bishop, *Neural Networks for Pattern Recognition.* Clarendon Press, 1995.

[9] C. M. Bishop, “Bayesian PCA,” in *Advances in Neural Information Processing Systems 11, NIPS*98*, (Denver, Colorado, USA, Nov. 30–Dec. 5, 1998), pp. 382–388, The MIT Press, 1999.

[10] C. M. Bishop, M. Svensén, and C. K. I. Williams, “GTM: The generative topographic mapping,” *Neural Computation*, vol. 10, no. 1, pp. 215–234, 1998.

[11] G. Boole, *An Investigation of the Laws of Thought.* Walton and Maberley, 1854.

[12] T. Briegel and V. Tresp, “Fisher scoring and a mixture of modes approach for approximate inference and learning in nonlinear state space models,” in *Advances in Neural Information Processing Systems 11, NIPS*98*, (Denver, Colorado, USA, Nov. 30–Dec. 5, 1998), pp. 403–409, The MIT Press, 1999.

[13] G. Burel, “Blind separation of sources: A nonlinear neural algorithm,” *Neural Networks*, vol. 5, no. 6, pp. 937–947, 1992.

[14] J.-F. Cardoso, “Multidimensional independent component analysis,” in *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, ICASSP’98*, (Seattle, Washington, USA, May 12–15), pp. 1941–1944, 1998.

[15] G. J. Chaitin, “On the length of programs for computing finite binary sequences,” *Journal of the ACM*, vol. 13, no. 4, pp. 547–569, 1966.

[16] A. Cichocki, L. Zhang, S. Choi, and S. Amari, “Nonlinear dynamic independent component analysis using state-space and neural network models,” in *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation, ICA’99*, (Aussois, France, Jan. 11–15), pp. 99–104, 1999.

[17] P. Comon, “Independent component analysis — a new concept?,” *Signal Processing*, vol. 36, pp. 287–314, 1994.

[18] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* Wiley & Sons, 1991.

[19] R. T. Cox, “Probability, frequency and reasonable expectation,” *American Journal of Physics*, vol. 14, no. 1, pp. 1–13, 1946.

[20] G. Deco and W. Brauer, “Nonlinear higher-order statistical decorrelation by volume-conserving neural architecture,” *Neural Networks*, vol. 8, no. 4, pp. 525–535, 1995.

[21] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society (Series B)*, vol. 39, pp. 1–38, 1977.

[22] D. C. Dennet, *Consciousness Explained.* Little, Brown and Co., 1991.

[23] H. Dürer and T. Waschulzik, “ESyNN — a model to abstractly emulate synchronization in neural networks,” in *Proceedings of the Ninth International Conference on Artificial Neural Networks, ICANN’99*, (Edinburgh, UK, Sep. 7–10), pp. 791–796, 1999.

[24] R. Eckhorn, R. Bauer, W. Jordan, M. Brosch, W. Kruse, M. Munk, and H. J. Reitboeck, “Coherent oscillations: A mechanism of feature linking in the visual cortex? Multiple electrode and correlation analyses in the cat,” *Biological Cybernetics*, vol. 60, pp. 121–130, 1989.

[25] B. Everitt, ed., *An Introduction to Latent Variable Models.* Chapman and Hall, 1984.

[26] D. J. Felleman and D. C. V. Essen, “Distributed hierarchical processing in the primate cerebral cortex,” *Cerebral Cortex*, vol. 1, no. 1, pp. 1–47, 1991.

[27] W. T. Freeman, “The generic viewpoint assumption in a Bayesian framework,” in Knill and Richards [64], pp. 365–389, 1996.

[28] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian Data Analysis.* Chapman & Hall, 1995.

[29] Z. Ghahramani and G. E. Hinton, “Hierarchical non-linear factor analysis and topographic maps,” in *Advances in Neural Information Processing Systems 10, NIPS*97*, (Denver, Colorado, USA, Dec. 1–6, 1997), pp. 486–492, The MIT Press, 1998.

[30] Z. Ghahramani and G. E. Hinton, “Variational learning for switching state-space models,” *Neural Computation*, vol. 12, no. 4, pp. 963–996, 2000.

[31] Z. Ghahramani and S. T. Roweis, “Learning nonlinear dynamical systems using an EM algorithm,” in *Advances in Neural Information Processing Systems 11, NIPS*98*, (Denver, Colorado, USA, Nov. 30–Dec. 5, 1998), pp. 599–605, The MIT Press, 1999.

[32] D. C. Gilbert, “Circuitry, architecture, and functional dynamics of visual cortex,” *Cerebral Cortex*, vol. 3, no. 5, pp. 373–386, 1993.

[33] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, eds., *Markov Chain Monte Carlo in Practice.* Chapman & Hall, 1996.

[34] M. Girolami, *Self-Organising Neural Networks — Independent Component Analysis and Blind Source Separation.* Springer-Verlag, 1999.

[35] R. L. Gorsuch, *Factor Analysis*. Lawrence Earlbaum Associates, 2nd ed., 1983.

[36] C. M. Gray, "Synchronous oscillations in neuronal systems, mechanisms and functions," *Journal of Computational Neuroscience*, vol. 1, pp. 11–39, 1994.

[37] C. M. Gray and W. Singer, "Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex," *Proc. Natl. Acad. Sci*, vol. 86, pp. 1698–1702, 1989.

[38] M. S. Grewal and A. P. Andrews, *Kalman Filtering*. Prentice-Hall, 1993.

[39] S. Haykin, *Neural Networks — A Comprehensive Foundation*. Prentice Hall, 2nd ed., 1998.

[40] R. Hecht-Nielsen, "Replicator neural networks for universal optimal source coding," *Science*, vol. 269, pp. 1860–1863, 1995.

[41] R. Herken, ed., *The Universal Turing Machine: a Half-Century Survey*. Oxford University Press, 1988.

[42] M. Herrmann and H. H. Yang, "Perspectives and limitations of self-organising maps in blind separation of source signals," in *Progress in Neural Information Processing, Proc. ICONIP'96*, (Wan Chai, Hong Kong, Sep. 24–27), pp. 1211–1216, Springer-Verlag, 1996.

[43] G. E. Hinton and T. J. Sejnowski, eds., *Unsupervised Learning: Foundations of Neural Computation*. Computational Neuroscience Series, The MIT Press, 1999.

[44] G. E. Hinton and D. van Camp, "Keeping neural networks simple by minimizing the description length of the weights," in *Proceedings of the COLT'93*, (Santa Cruz, California, USA, July 26–28), pp. 5–13, 1993.

[45] S. Hochreiter and M. C. Mozer, "An electric field approach to independent component analysis," in *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, (Helsinki, Finland, June 19–22), pp. 45–50, 2000.

[46] S. Hochreiter and J. Schmidhuber, "Flat minima," *Neural Computation*, vol. 9, no. 1, pp. 1–42, 1997.

[47] S. Hochreiter and J. Schmidhuber, "Feature extraction through LOCOCODE," *Neural Computation*, vol. 11, no. 3, pp. 679–714, 1999.

[48] S. Hochreiter and J. Schmidhuber, "LOCOCODE performs nonlinear ICA without knowing the number of sources," in *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation, ICA'99*, (Aussois, France, Jan. 11–15), pp. 149–154, 1999.

[49] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[50] J.-M. Hupé, A. C. J. B. R. Payne, , S. G. Lomber, P. Girard, and J. Bullier, "Cortical feedback improves discrimination between figure and background by v1, v2 and v3 neurons," *Nature*, vol. 394, pp. 784–787, 1998.

[51] A. Hyvärinen, "Fast and robust fixed-point algorithms for independent component analysis," *IEEE Transactions on Neural Networks*, vol. 10, no. 3, pp. 626–634, 1999.

[52] A. Hyvärinen, "Survey on independent component analysis," *Neural Computing Surveys*, vol. 2, pp. 94–128, 1999.

[53] A. Hyvärinen and P. O. Hoyer, "Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces," *Neural Computation*, vol. 12, no. 7, pp. 1705–1720, 2000.

[54] A. Hyvärinen and P. O. Hoyer, "Emergence of topography and complex cell properties from natural images using extensions of ICA," in *Advances in Neural Information Processing Systems 12, NIPS*99*, (Denver, Colorado, USA, Nov. 29 – Dec. 4, 1999), pp. 827–833, The MIT Press, 2000.

[55] A. Hyvärinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural Computation*, vol. 9, no. 7, pp. 1483–1492, 1997.

[56] A. Hyvärinen, J. Särelä, and R. Vigário, "Bumps and spikes: Artifacts generated by independent component analysis with insufficient sample size," in *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation, ICA'99*, (Aussois, France, Jan. 11–15), pp. 425–429, 1999.

[57] E. T. Jaynes, "Probability theory: The logic of science." Available from `http://bayes.wustl.edu/etj/prob.html`, 1996.

[58] I. T. Jolliffe, *Principal Component Analysis*. Springer-Verlag, 1986.

[59] M. I. Jordan, ed., *Learning in Graphical Models*. The MIT Press, 1999.

[60] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," in Jordan [59], pp. 105–161, 1999.

[61] C. Jutten and J. Herault, "Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture," *Signal Processing*, vol. 24, pp. 1–10, 1991.

[62] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, eds., *Principles of Neural Science*. Elsevier, 3rd ed., 1991.

[63] M. Kendall, *Multivariate Analysis*. Charles Griffin & Co., 1975.

[64] D. C. Knill and W. Richards, eds., *Perception as Bayesian Inference*. Cambridge University Press, 1996.

[65] T. Kohonen, *Self-Organizing Maps*. Springer-Verlag, 2nd, extended ed., 1997.

[66] T. Kohonen, S. Kaski, and H. Lappalainen, "Self-organized formation of various invariant-feature filters in the Adaptive-Subspace SOM," *Neural Computation*, vol. 9, no. 6, pp. 1321–1344, 1997.

[67] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Problems of Information Transmission*, vol. 1, pp. 1–17, 1965. Translated from Problemy Peredachi Informatsii (in Russian).

[68] S. M. Kosslyn, W. L. Thompson, I. J. Kim, and N. M. Alpert, "Topographical representations of mental images in primary visual cortex," *Nature*, vol. 378, pp. 496–498, 1995.

[69] S. W. Kuffler, J. G. Nicholls, and A. R. Martin, *From Neuron to Brain.* Sinauer Associates Inc. Publishers, 2nd ed., 1984.

[70] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951.

[71] P. S. Laplace, "Mémoire sur la probabilité des causes par les événements," *Mémoires de l'Académie Royale des Sciences*, vol. 6, pp. 621–656, 1774. English translation in [123].

[72] H. Lappalainen, "Fast fixed-point algorithms for Bayesian blind source separation," Publications in Computer and Information Science A56, Helsinki University of Technology, Espoo, Finland, 1999.

[73] S. Lauritzen, ed., *Graphical Models.* Oxford University Press, 1996.

[74] D. D. Lee and H. S. Seung, "Unsupervised learning by convex and conic coding," in *Advances in Neural Information Processing Systems 9, NIPS*96*, (Denver, Colorado, USA, Nov. 2–5, 1996), pp. 515–521, The MIT Press, 1997.

[75] P. M. Lee, *Bayesian Statistics: An Introduction.* Oxford University Press, 1989.

[76] T.-W. Lee, *Independent Component Analysis — Theory and Applications.* Kluwer, 1998.

[77] L. A. Levin, "Universal sequential search problems," *Problems of Information Transmission*, vol. 9, no. 3, pp. 256–266, 1973.

[78] M. Li and P. M. B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, 2nd, extended ed., 1997.

[79] J. K. Lin, D. Grier, and J. D. Cowan, "Faithful representation of separable input distribution," *Neural Computation*, vol. 9, no. 6, pp. 1305–1320, 1997.

[80] W. Maass and C. M. Bishop, eds., *Pulsed Neural Networks.* The MIT Press, 1999.

[81] D. J. C. MacKay, "A practical Bayesian framework for backpropagation networks," *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[82] D. J. C. MacKay, "Developments in probabilistic modelling with neural networks—ensemble learning," in *Neural Networks: Artificial Intelligence and Industrial Applications. Proceedings of the 3rd Annual Symposium on Neural Networks*, (Nijmegen, Netherlands, Sep. 14–15), pp. 191–198, Springer-Verlag, 1995.

[83] D. J. C. MacKay, "Ensemble learning for hidden Markov models." Available from `http://wol.ra.phy.cam.ac.uk/`, 1997.

[84] D. J. C. MacKay, "Choice of basis for laplace approximation," *Machine Learning*, vol. 33, no. 1, pp. 77–86, 1998.

[85] D. J. C. MacKay and M. N. Gibbs, "Density networks," in *Proceedings of Society for General Microbiology Edinburgh Meeting*, 1997.

[86] G. C. Marques and L. B. Almeida, "An objective function for independence," in *Proceedings of the International Conference on Neural Networks, ICNN'96*, (Washington, DC, USA, June 3–6), pp. 453–457, 1996.

[87] G. C. Marques and L. B. Almeida, "Separation of nonlinear mixtures using pattern repulsion," in *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation, ICA'99*, (Aussois, France, Jan. 11–15), pp. 277–282, 1999.

[88] P. S. Maybeck, *Stochastic Models, Estimation, and Control*, vol. 1. Academic Press, 1979.

[89] G. J. McLachlan and K. E. Basford, *Mixture Models. Inference and Applications to Clustering.* Marcel Dekker, 1988.

[90] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, no. 2, pp. 281–294, 1989.

[91] R. M. Neal, "Connectionist learning of belief networks," *Artificial Intelligence*, vol. 56, no. 1, pp. 71–113, 1992.

[92] R. M. Neal, *Bayesian Learning for Neural Networks.* No. 118 in Lecture Notes in Statistics, Springer-Verlag, 1996.

[93] R. M. Neal and G. E. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants," in Jordan [59], pp. 355–368, 1999.

[94] J.-H. Oh and H. S. Seung, "Learning generative models with the up-propagation algorithm," in *Advances in Neural Information Processing Systems 10, NIPS*97*, (Denver, Colorado, USA, Dec. 1–6, 1997), pp. 605–611, The MIT Press, 1998.

[95] E. Oja, "The nonlinear PCA learning rule in independent component analysis," *Neurocomputing*, vol. 17, no. 1, pp. 25–46, 1997.

[96] J. J. Oliver and R. A. Baxter, "MML and Bayesianism: Similarities and differences," Tech. Rep. TR 206, Department of Computer Science, Monash University, Australia, 1994.

[97] J. J. Oliver and D. J. Hand, "Introduction to minimum encoding inference," Tech. Rep. TR 205, Department of Computer Science, Monash University, Australia, 1994.

[98] P. Pajunen, "Nonlinear independent component analysis by self-organizing maps," in *Proceedings of the Sixth International Conference on Artificial Neural Networks, ICANN'96*, (Bochum, Germany, July 16–19), pp. 815–819, 1996.

[99] P. Pajunen, "Blind source separation using algorithmic information theory," *Neurocomputing*, vol. 22, pp. 35–48, 1998.

[100] P. Pajunen and J. Karhunen, "A maximum likelihood approach to nonlinear blind source separation," in *Proceedings of the Seventh International Conference on Artificial Neural Networks, ICANN'97*, (Lausanne, Switzerland, Oct. 8–10), pp. 541–546, 1997.

[101] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufman, 1988.

[102] J. W. Pratt, H. Raiffa, and R. O. Schlaifer, *Introduction to Statistical Decision Theory*. The MIT Press, 1995.

[103] S. J. Press, *Bayesian Statistics: Principles, Models, and Applications*. Wiley, 1989.

[104] P. Rakic and W. Singer, eds., *Neurobiology of Neocortex*. John Wiley & Sons, 1988.

[105] R. P. N. Rao and D. H. Ballard, "Kalman filter model of the visual cortex," *Neural Computation*, vol. 9, no. 4, pp. 721–763, 1997.

[106] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978.

[107] J. Rissanen, "Fisher information and stochastic complexity," *IEEE Transactions on Information Theory*, vol. 42, no. 1, pp. 40–47, 1996.

[108] J. Rissanen and G. G. Langdon, Jr., "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979.

[109] J. Rissanen and G. G. Langdon, Jr., "Universal modeling and coding," *IEEE Transactions on Information Theory*, vol. 27, pp. 12–23, 1981.

[110] D. Rubin and D. Thayer, "EM algorithms for factor analysis," *Psychometrika*, vol. 47, pp. 69–76, 1982.

[111] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error backpropagation," in *Parallel distributed processing* (D. E. Rumelhart and J. L. McClelland, eds.), vol. 1, pp. 318–362, The MIT Press, 1986.

[112] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[113] L. K. Saul, T. Jaakkola, and M. I. Jordan, "Mean field theory for sigmoid belief networks," *Journal of Artificial Intelligence Research*, vol. 4, pp. 61–76, 1996.

[114] L. J. Savage, *The Foundations of Statistics*. Dover Publications, 1954.

[115] M. J. Schervish, *Theory of Statistics*. Springer-Verlag, 1995.

[116] J. Schmidhuber, "Discovering neural nets with low Kolmogorov complexity and high generalization capability," *Neural Networks*, vol. 10, no. 5, pp. 857–873, 1997.

[117] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, 1948.

[118] R. H. Shumway and D. S. Stoffer, "An approach to time series smoothing and forecasting using the EM algorithm," *Journal of Time Series Analysis*, vol. 3, no. 4, pp. 253–264, 1982.

[119] R. J. Solomonoff, "A formal theory of inductive inference. Part I," *Information and Control*, vol. 7, no. 1, pp. 1–22, 1964.

[120] R. J. Solomonoff, "A formal theory of inductive inference. Part II," *Information and Control*, vol. 7, no. 2, pp. 224–254, 1964.

[121] H. W. Sorenson, ed., *Kalman Filtering: Theory and Application*. IEEE Press, 1985.

[122] C. Spearman, ""General intelligence," objectively determined and measured," *American Journal of Psychology*, vol. 15, pp. 201–293, 1904.

[123] S. M. Stigler, "Translation of Laplace's 1774 memoir on "Probability of causes"," *Statistical Science*, vol. 1, no. 3, pp. 359–378, 1986.

[124] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[125] A. Taleb and C. Jutten, "Nonlinear source separation: The post-nonlinear mixtures," in *Proceedings of the European Symposium on Artificial Neural Networks, ESANN'97*, (Bruges, Belgium, Apr. 16–18), pp. 279–284, 1997.

[126] A. Taleb and C. Jutten, "Source separation in post-nonlinear mixtures," *IEEE Transactions on Signal Processing*, vol. 47, no. 10, pp. 2807–2820, 1999.

[127] K. Tanaka, "Inferotemporal cortex and object vision," *Annual Reviews in Neuroscience*, vol. 10, pp. 109–139, 1996.

[128] H. Valpola, X. Giannakopoulos, A. Honkela, and J. Karhunen, "Nonlinear independent component analysis using ensemble learning: Experiments and discussion," in *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, (Helsinki, Finland, June 19–22), pp. 351–356, 2000.

[129] A. Wald, *Statistical Decision Functions*. Wiley, 1950.

[130] C. S. Wallace and D. M. Boulton, "An information measure for classification," *Computer Journal*, vol. 11, no. 2, pp. 185–194, 1968.

[131] C. S. Wallace and P. R. Freeman, "Estimation and inference by compact coding," *Journal of the Royal Statistical Society (Series B)*, vol. 49, no. 3, pp. 240–265, 1987.

[132] J. E. Whitesitt, *Boolean Algebra and Its Applications*. Dover Publications, 1995.

[133] R. R. Yager and L. A. Zadeh, *An Introduction to Fuzzy Logic Applications in Intelligent Systems*. Kluwer Academic Publishers, 1992.

[134] H. H. Yang, S. Amari, and A. Cichocki, "Information back-propagation for blind separation of sources from non-linear mixtures," in *Proceedings of the International Conference on Neural Networks, ICNN'97*, (Houston, Texas, USA, June 9–12), 1997.

[135] H. H. Yang, S. Amari, and A. Cichocki, "Information-theoretic approach to blind separation of sources in non-linear mixture," *Signal Processing*, vol. 64, pp. 291–300, 1998.

[136] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.

USING AN MDL-BASED COST FUNCTION WITH NEURAL NETWORKS

Harri Lappalainen

In *Proceedings of the International Joint Conference on Neural Networks, IJCNN'98*, (Anchorage, Alaska, USA, May 4–9), pp. 2384-2389, 1998.

Abstract

The minimum description length (MDL) principle is an information theoretically based method to learn models from data. This paper presents how to efficiently use an MDL-based cost function with neural networks. As usual, the cost function can be used to adapt the parameters in the network, but it can also include terms to measure the complexity of the structure of the network and can thus be applied to determine the optimal structure. The basic idea is to convert a conventional neural network such that each parameter and each output of the neurons is assigned a mean and a variance. This greatly simplifies the computation of the description length and its gradient with respect to the parameters, which can then be adapted using standard gradient descent.

1 Introduction

Intuitively, the idea behind MDL is that in order to be able to compress any given data one has to capture its regularities and structure. There are information-theoretical reasons why minimising the description length should yield models with good generalisation properties [5], and it has been shown in practise that description length has strong correlation with the expected classification error of an independent test set [3].

To understand the principle behind MDL, suppose there is a sender who wants to transmit a given data to a receiver. We shall assume that the sender will use a two-part coding scheme: he will first send a model of the data and then the data which is compressed using the model. The more bandwidth is used for the model the more the data is compressed. At some point, however, the cost of using a longer model becomes higher than the gain in compression of the data. Clearly, the length of the model should not exceed the length of the description of the data without using any model. Figure 1 shows schematically the relation between model complexity and the description length of the model and the data. Of course it is not necessary to actually code and transmit the model and the data in order to estimate the optimal model. It is enough to calculate what would be the description length if the data were compressed.

The terms under- and over-fitting can be seen from MDL point of view. The optimal model is by our definition the one that has the minimum description length. Under-fitting means that the complexity of a model is below, and over-fitting that the complexity is above that of the optimal model.

It is important to recognise that the optimal model is not unique. Before transmitting the model and the data, the sender and the receiver must have agreed on how the message is coded, or otherwise the receiver would not know how to interpret it. The description length of the information used in the interpretation cannot be measured, because we would

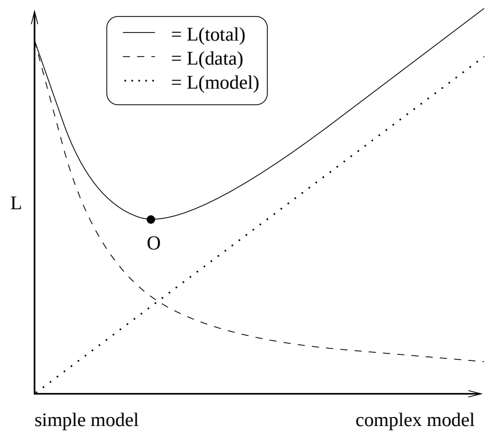

Figure 1: The relation between model complexity and description length is shown schematically. Increasing model complexity allows better coding of data, but with the cost of increasing the length of model description. The best model has the minimum total description length (point O).

need to know how to interpret the coded interpretation, etc. Therefore there must always be some aspect of the model whose description length is neglected, namely the coding scheme. It determines the description length of the data, which means that models can only be compared with respect to a given coding scheme. Using Bayesian terms, the coding scheme implicitly defines a prior over the models. It is, however, often much easier to incorporate prior knowledge in the design of the code than assign a Bayesian prior over the models.

1.1 MDL-based cost functions for supervised and unsupervised learning

A common problem in pattern recognition is that one wants to recognise objects in a given raw data. One might have a large amount of raw data available, but only a small amount of labelled data. The typical approach is to extract features from the raw data in an unsupervised manner and then to find a mapping from the features to the corresponding labels (desired outputs) in a supervised manner. The whole process can be described by $I_U \to (F_U = I_S) \to D_S$, where the arrows stands for mappings, I for input, F for feature, D for desired output, U for unsupervised and S for supervised.

Let us denote the model by M. In supervised learning one wants to find a mapping from the input data I_S to the desired output data D_S. The mapping is described by the model M_S, whose structure and parameters are to be estimated. In a conventional neural network, the cost function C measures the error in the mapping: $C(D_S - M_S(I_S))$. The MDL-based cost function for supervised learning is $L(D_S|M_S, I_S)+L(M_S)$: the description length of the output data given the model and the input data plus the description length of the model. The first term corresponds to $C(D_S - M_S(I_S))$ and the second term penalises for the complexity of the model. The input is assumed to be known by both the sender and the receiver and is not included in the description length.

In unsupervised learning one wants to find features F_U which compactly describe the input data I_U. The quality of the features is evaluated by estimating how much information about inputs they preserve, that is, how well the inputs can be reconstructed from the features. We shall consider a generative model M_U which defines the *reconstruction* mapping: $\hat{I}_U = M_U(F_U)$. The reconstruction error which measures the quality of the features is then $C(I_U - M_U(F_U))$. We shall use vector quantisation to illuminate the use of a generative model.

In vector quantisation, the model consists of model vectors $\boldsymbol{m}_i$, which directly define the reconstruction mapping M_U: given an index i, the reconstruction is $\boldsymbol{m}_i$. The index i thus plays the role of the feature. The quality of the feature is measured with the reconstruction error C, or quantisation error as it is often called in connection with vector quantisation. The feature is defined as the index that minimises the reconstruction error, or to put it in other words, the index of the closest model vector: $F_U = M_U^{-1}(I_U) = \arg\min_F C(I_U - M_U(F))$. The model can be adapted by minimising the reconstruction error, that is, by moving the closest model vector even closer to the input. The features and the model are thus both defined as the minimum of the reconstruction error.

In addition to the term measuring the reconstruction error, the MDL-based cost function for unsupervised learning has terms penalising the complexity of the model and the features: $L(I_U|M_U, F_U) + L(M_U) + L(F_U)$. The cost function is used in exactly the same way as with vector quantisation: the features and the model are found by minimising it.

The features are often considered a subset of the parameters of the model, because they too have to be estimated from the data and are included in the cost function. The only difference between the features and the parameters is that the features change for each input element, whereas the parameters are constant for different data elements.

2 Measuring the description length

Typically neural networks have relatively simple structure but many parameters, and it is usually easy to design a satisfactory coding for the structure. We shall therefore concentrate on measuring the description length of the parameters of the model.

If the transmitted data consists of discrete elements with known probabilities, then the length needed in transmission can be computed. It is well-known that if the probability of a data element D is $P(D)$, then the transmission of the element D using the optimal code takes $-\log_2 P(D)$ bits [9, 8]. We shall hereafter measure information in base e instead of base 2: $-\log_2 P(D)$ bits equals $-\ln P(D)$ nats, where 1 nat equals $\log_2 e$ bits.

The output data and the parameters in the network have continuous values instead of discrete. Communicating a random continuous value at an infinite precision would require an infinite number of bits, and therefore the values have to be discretised. If the discretisation is done with precision ϵ_x, then the range of values is divided into small bins with size ϵ_x. Suppose we know the probability distribution $p(x)$. If ϵ_x is small enough, $p(x)$ is approximately linear throughout each bin and the probability of each bin is approximately $p(x)\epsilon_x$, which means that the description length L of the value x is

$$L(x) = -\ln p(x)\epsilon_x = -\ln p(x) - \ln \epsilon_x. \tag{1}$$

Equation 1 shows that in order to compute the description length, *each continuos value must be assigned a probability distribution and an accuracy.*

The model is described by its structure, its parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots]$ and their accuracies ϵ . We shall consider a coding scheme where the model is used to assign a probability distribution $p(D)$ for each data element $D \in \mathcal{D}$. Some of the parameters may also be hyper parameters which are used to assign probability distribution p_{θ_i} for other parameters. The data is discretised with accuracy $\epsilon_{\mathcal{D}}$, but that can be considered an integral part of the coding scheme. This is because the effect of $\epsilon_{\mathcal{D}}$ does not depend on the model: it always adds a constant $-N \ln \epsilon_{\mathcal{D}}$ term in the description length, and can therefore be omitted (N is the number of data elements). The accuracy of the parameters (ϵ) does depend on the model, however, and must be taken into account.

In the literature, several different ways to measure the description length have been proposed. Our approach dates back to Wallace and Boulton [10] and similar approaches have been used in [4, 11]. Alternative approaches have been presented e.g. in [6, 7].

Suppose the target value for the parameter vector is $\boldsymbol{\theta}$. Let us denote the actual discretised value by $\hat{\boldsymbol{\theta}}$ and the corresponding round-off error by $\tilde{\boldsymbol{\theta}} = \hat{\boldsymbol{\theta}} - \boldsymbol{\theta}$. The errors depend on the locations of the borders between the bins, but there is no reason to favour one discretisation over the others, and therefore we should compute the *expectation* of the description length over the distribution of $\tilde{\boldsymbol{\theta}}$, which is approximately even in the range $[-\epsilon\ /2, \epsilon\ /2]$. We are also going to assume that the errors are uncorrelated, that is, $E\{\tilde{\theta}_i\tilde{\theta}_j\} = 0$.

The expected description length can be approximated by utilising the second order Taylor's series expansion of $L(\hat{\boldsymbol{\theta}})$ about $\boldsymbol{\theta}$.

$$L(\hat{\boldsymbol{\theta}}) \approx L(\boldsymbol{\theta}) + \sum_i \frac{\partial L}{\partial \hat{\theta}_i}\tilde{\theta}_i + \frac{1}{2}\sum_i \sum_j \frac{\partial^2 L}{\partial \hat{\theta}_i \partial \hat{\theta}_j}\tilde{\theta}_i\tilde{\theta}_j \tag{2}$$

Here the derivatives of L are evaluated at $\boldsymbol{\theta}$. Taking expectation over $\tilde{\boldsymbol{\theta}}$ yields the cost function:

$$L_E(\boldsymbol{\theta}, \epsilon\ ) \overset{def}{=} E_{\tilde{\ }}\{L(\hat{\boldsymbol{\theta}})\} \approx L(\boldsymbol{\theta}) + \frac{1}{2}\sum_i \frac{\partial^2 L}{\partial \hat{\theta}_i^2} E\{\tilde{\theta}_i^2\}. \tag{3}$$

Here we have used $E\{\tilde{\theta}_i\} = 0$ for all i and $E\{\tilde{\theta}_i\tilde{\theta}_j\} = 0$ for all $i \neq j$. The optimal parameters $\boldsymbol{\theta}$ and their accuracies ϵ can be solved by minimising the expected description length L_E.

Since the cost function includes second order derivatives of L with respect to the parameters, it might be troublesome to use it with a large neural network. First the outputs have to be computed in a feedforward phase as usual. Approximations of the second order derivatives needed in the cost function can be computed in a process analogous to backpropagation. It involves two feedback phases: the first one is used to compute the first order derivatives and the second one to compute the second order derivatives. It is then laborious to use backpropagation to compute the gradient, because it has to be applied to all three phases, resulting in a total of six phases.

Fortunately we can overcome these difficulties by recognising that it is not the second order derivatives we wish to evaluate but the expectation over $\tilde{\boldsymbol{\theta}}$.

3 Computing the expectation of the description length for neural networks

In this section we shall describe a method for computing the expectation of the description length for neural networks in an efficient feedforward process. We shall first write down the feedforward equations for the outputs of the neurons and the description length, and then take expectations of the equations. This results in converted feedforward equations, which automatically yield the expected description length. It is then easy to use backpropagation to adapt the parameters. For the sake of simplicity, we shall only consider supervised learning with multilayer perceptrons (MLP), although the method is general and can be applied to practically any type of neural network, both supervised and unsupervised learning.

3.1 Feedforward equations

We shall consider a standard MLP-network with input, hidden and output layers. In order to be able to write the feedforward equations in a compact form, we shall assign all parameters and outputs of the neurons a unique index. In our notation, ξ_i can mean either the value of a parameter or output of a neuron, that is, ξ_i are used to denote any value than can be an input for neurons. The set of indices for the parameters is denoted by $\mathcal{P}$ and the transfer functions of the neurons are denoted by f_i. The values ξ_i are defined by equation 4.

$$\xi_i(\hat{\boldsymbol{\theta}}, t) \overset{def}{=} \begin{cases} \hat{\theta}_i = \theta_i + \tilde{\theta}_i & \text{parameters } (i \in \mathcal{P}) \\ D_i(t) & \text{desired outputs} \\ I_i(t) & \text{input neurons} \\ f_i(\xi_j | j \in \mathcal{J}_i) & \text{other neurons} \end{cases} \tag{4}$$

Input and output data is parametrised by time t. We have used a shorthand notation for the parameters of a function. For example $f(\xi_j | j \in \{2,4,5\}) = f(\xi_2, \xi_4, \xi_5)$. The set $\mathcal{J}_i$ thus contains the indices of the inputs to the neuron i or, equivalently, indices of the parameters of the function f_i. The network is assumed to be strictly feedforward, which means that $j \in \mathcal{J}_i$ implies $j < i$.

For hidden and output neurons the transfer functions f_i are like in any conventional neural network. They can be sums of inputs multiplied by weights, sigmoids, radial basis functions, etc.

The cost function for supervised learning is $L(M_S) + L(D_S|M_S)$ as explained in section 1.1. The description lengths are computed according to equation 1, with the exception that the terms $-\ln \epsilon$ are omitted from $L(D_S|M_S)$. For each D_i we shall assign a function f_j, which is used to compute the terms $-\ln p(D_i)$. The set of indices for these functions is denoted by $\mathcal{L}_\mathcal{D}$. Similarly, the set $\mathcal{L}_\mathcal{P}$ comprises of the indices of functions f_j, which evaluate the terms $-\ln p(\theta_i)$. We can now write down the cost function in terms of ξ_i and ϵ_{θ_i}.

$$L(\hat{\boldsymbol{\theta}}, \boldsymbol{\epsilon}\ ) = \sum_{i \in \mathcal{L}_\mathcal{P}} \xi_i - \sum_{i \in \mathcal{P}} \ln \epsilon_{\theta_i} + \sum_{t=1}^{N} \sum_{i \in \mathcal{L}_\mathcal{D}} \xi_i(t) \tag{5}$$

The first two terms correspond to $L(M_S)$ and the third term to $L(D_S|M_S)$.

If a parameter θ_i does not have an associated neuron in the set $\mathcal{L}_\mathcal{P}$, it means that we tacitly assume the probability distribution $p(\theta_i)$ to be constant throughout the range of values of θ_i, that is, we assume the value of the parameter to be evenly distributed. It has to be reminded that although the constant term $-\ln p(\theta_i)$ can be omitted when adapting the parameters and their accuracies, it should still be taken into account when models with different parametrisations are compared.

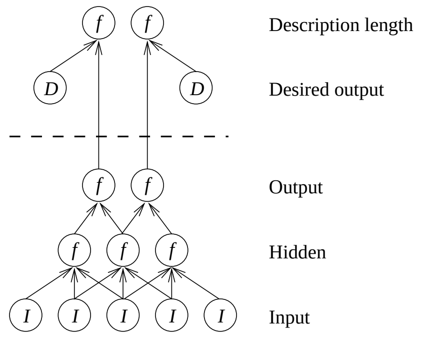

Figure 2: Structure of a MLP network with an MDL-based cost function is shown schematically. The layers below the dotted line are the same as in a conventional MLP: input, hidden and output layers. The functions above the dotted line are used to compute the cost function L.

The structure of the network is shown in figure 2. Desired outputs are marked by D, input neurons by I, and other neurons by f. The parameters of the network are not shown. The functions f above the dotted line are the ones used to compute the description length of the parameters and the data.

3.2 Expectation of the description length

Let us define $\mu_i \overset{def}{=} E_{\tilde{\boldsymbol{\theta}}}\{\xi_i\}$. Taking expectation over $\tilde{\boldsymbol{\theta}}$ from equation 4 yields

$$\mu_i(\boldsymbol{\theta}, \boldsymbol{\epsilon}\ , t) = \begin{cases} \theta_i \\ D_i(t) \\ I_i(t) \\ E\{f_i(\xi_j | j \in \mathcal{J}_i)\} \end{cases} \tag{6}$$

and expectation from equation 5 yields

$$L_E(\boldsymbol{\theta}, \boldsymbol{\epsilon}\ ) \overset{def}{=} E\{L(\hat{\boldsymbol{\theta}}, \boldsymbol{\epsilon}\ )\} = \sum_{i \in \mathcal{L}_\mathcal{P}} \mu_i - \sum_{i \in \mathcal{P}} \ln \epsilon_{\theta_i} + \sum_{t=1}^{N} \sum_{i \in \mathcal{L}_\mathcal{D}} \mu_i(t). \tag{7}$$

We have dropped the subscripts from the expectation operators, but they are always taken over $\tilde{\boldsymbol{\theta}}$.

3.3 Expectation of a function

Equations 6 and 7 show that we could compute the expected description length if we knew how to evaluate the expectations of the functions f_i.

Let $f_i(\xi_j | j \in \mathcal{J}_i)$ be a function whose expectation value we would like to compute. As before, we shall approximate f_i with its second order Taylor's series expansion. However, we are not going to expand f_i with respect to the discretised parameters $\hat{\boldsymbol{\theta}}$ of the network but the direct parameters ξ_j of the function f_i. The expansion is thus computed about the expectation values $\mu_j = E\{\xi_j\}$.

$$\xi_i = f_i(\xi_j | j \in \mathcal{J}_i) \approx f_i(\mu_j | j \in \mathcal{J}_i) + \sum_{j \in \mathcal{J}_i} \frac{\partial f_i}{\partial \xi_j}(\xi_j - \mu_j) + \frac{1}{2} \sum_{j \in \mathcal{J}_i} \sum_{k \in \mathcal{J}_i} \frac{\partial^2 f_i}{\partial \xi_j \partial \xi_k}(\xi_j - \mu_j)(\xi_k - \mu_k) \quad (8)$$

Taking expectation from both sides of equation 8 yields

$$\mu_i = E\{f_i(\xi_j | j \in \mathcal{J}_i)\} \approx f_i(\mu_j | j \in \mathcal{J}_i) + \sum_{j \in \mathcal{J}_i} \frac{v_j}{2} \frac{\partial^2 f_i}{\partial {\xi_j}^2}. \quad (9)$$

Here we have denoted the variance of ξ_j by v_j: $v_j \overset{def}{=} E\{(\xi_j - \mu_j)^2\}$. The first order terms disappear because $E\{\xi_j - \mu_j\} = 0$. We have also assumed that for all $j \neq k$, either ξ_j and ξ_k are uncorrelated or $\frac{\partial^2 f_i}{\partial \xi_j \partial \xi_k} = 0$, which removes the second order cross terms.

3.4 Variance of a function

We still need the variances v_j. Since each error $\tilde{\theta}_i$ is evenly distributed in the range $[-\epsilon_{\theta_i}/2, \epsilon_{\theta_i}/2]$, the variances of the parameters are given by $v_i = \epsilon^2_{\theta_i}/12$. The variance of the inputs and desired outputs can be assumed to be zero, or they can be assigned some values if there is prior knowledge about, for example, the equipment used to measure the values.

In order to compute the variance of the function f_i, we shall approximate it with first order Taylor's series expansion.

$$\xi_i = f_i(\xi_j | j \in \mathcal{J}_i) \approx f_i(\mu_j | j \in \mathcal{J}_i) + \sum_{j \in \mathcal{J}_i} \frac{\partial f_i}{\partial \xi_j}(\xi_j - \mu_j) \quad (10)$$

According to this approximation, $\mu_i = E\{\xi_i\} \approx f_i(\mu_j | j \in \mathcal{J}_i)$, which yields

$$v_i = E\{(\xi_i - \mu_i)^2\} \approx \sum_{j \in \mathcal{J}_i} \sum_{k \in \mathcal{J}_i} \frac{\partial f_i}{\partial \xi_j} \frac{\partial f_i}{\partial \xi_k} E\{(\xi_j - \mu_j)(\xi_k - \mu_k)\} \quad (11)$$

Again we can drop out the cross terms if all ξ_j are mutually uncorrelated. This yields our final approximation for the variance of a function:

$$v_i \approx \sum_{j \in \mathcal{J}_i} \left(\frac{\partial f_i}{\partial \xi_j}\right)^2 v_j \quad (12)$$

Notice that when computing the variance, one cannot mix the first and second order approximation for μ_i, since that might result in negative values for v_i.

3.5 Converted feedforward equations

We shall now collect together the results derived in this section.

$$\mu_i(\boldsymbol{\theta}, \boldsymbol{\sigma^2}, t) = \begin{cases} \theta_i \\ D_i(t) \\ I_i(t) \\ f_i(\mu_j | j \in \mathcal{J}_i) + \sum_{j \in \mathcal{J}_i} \frac{v_j}{2} \frac{\partial^2 f_i}{\partial {\xi_j}^2} \end{cases} \quad (13)$$

$$v_i(\boldsymbol{\theta}, \boldsymbol{\sigma^2}, t) = \begin{cases} \sigma_i^2 = \epsilon^2_{\theta_i}/12 \\ 0 \text{ (or the variance of } D_i(t)) \\ 0 \text{ (or the variance of } I_i(t)) \\ \sum_{j \in \mathcal{J}_i} \left(\frac{\partial f_i}{\partial \xi_j}\right)^2 v_j \end{cases} \quad (14)$$

Substituting $\epsilon_{\theta_i} = (12 v_i)^{1/2}$ into equation 7 yields equation 15 for the expected description length L_E.

$$L_E(\boldsymbol{\theta}, \boldsymbol{\sigma^2}) = \sum_{i \in \mathcal{L_P}} \mu_i - \sum_{i \in \mathcal{P}} \frac{1}{2} \ln 12 v_i + \sum_{t=1}^{N} \sum_{i \in \mathcal{L_D}} \mu_i(t) \quad (15)$$

Equations 13 and 14 describe how to convert a standard feedforward network into a network where the output of each neuron is assigned a mean and a variance. Equation 15 then defines the MDL-based cost function for such a network. The gradient of the cost function can be computed using standard backpropagation algorithm.

4 Discussion

4.1 Effects of approximations

In the approximation of the means and variances, the inputs of each function are assumed to be uncorrelated. If there exist i and j such that the value of θ_i is propagated in more than one separate route to the parameters of the function f_j, then the parameters are correlated and the computation of mean and variance may be inaccurate. This should not be a very severe restriction; in an MLP with one hidden layer, for example, no parameter (weight) affects any output neuron in more than one separate route. Should the approximation turn out to be too inaccurate, some of the cross terms may be taken into account.

We have assumed that the errors $\tilde{\boldsymbol{\theta}}$ made in the discretisation of the parameters are mutually uncorrelated. It can be argued that the estimate of the description length would be more accurate if we took into account the dependence between parameters: a change in the value of one parameter might be partially compensated by a suitable change in the values of the others. Assuming uncorrelated errors effectively penalises parametrisations with strong dependencies between parameters, since the description length could be made shorter using a parametrisation which removes the dependencies. Usually it is desirable to favour parametrisations with small dependencies, and thus the assumption of uncorrelated discretisation errors is reasonable.

In order to have a decodable message, the accuracies of the parameter values should be coded and sent before sending the truncated parameters. Wallace and Freeman [11] argue that the values and the accuracies of the parameters are not independent, and one can construct a decodable message with almost the same code length we have used.

4.2 Relation to previous work

If we neglect the effect of finite accuracy to the description length and assume an infinite precision, then the parameter values that minimise L are the ones that maximise the posterior probability of the parameters, that is, the maximum a posteriori estimate. They are not necessarily the same that minimise L when finite accuracy is taken into account, however. Taking the expectation over $\tilde{\boldsymbol{\theta}}$ effectively measures and penalises the sensitivity of L to the values of the parameters, thus finding a flatter minimum of L, which corresponds to more probability mass of the posterior density.

Using such MDL-based arguments, Hochreiter and Schmidhuber have arrived in a very similar algorithm, which they call the flat minimum search (FMS) [2]. Due to the similarity of the penalty for the complexity of the model, the promising results they have achieved should be reproductable with our method. Indeed, in preliminary simulations, we have been able to reproduce the experiment 1 in [2]: noisy classification.

Although the cost function in FMS is very similar to ours, it does not define a description length, and it is thus difficult to include measures of the complexity of the structure of the network. The computation of our cost function appears to be more simple, but, on the other hand, FMS does not assume independence of the parameters of the functions in the network. In FMS the user has to give an extra parameter, a tolerable error, which regulates the trade-off between the description length of the parameters and the data. In our approach, the optimal accuracies for the parameters are estimated directly from the data.

MDL has been used with neural networks to find representations in an unsupervised manner in [1, 12, 13]. The main focus therein has been on discrete valued features, such as the indices of vectors in vector quantisation. This paper concentrates on the coding of real valued parameters and features, and thus complements the previous work.

References

[1] Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and Helmholz free energy. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *NIPS 6*, pages 3–10, San Francisco, 1994. Morgan Kaufmann.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, January 1997.

[3] Bernhard Pfahringer. Compression-based feature subset selection. In P. Turney, editor, *IJCAI-95 Workshop on Data Engineering for Inductive Learning*. IJCAI-95 Workshop Program Working Notes, Montreal, Canada, 1995.

[4] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[5] Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11(2):416–431, 1983.

[6] Jorma Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society (Series B)*, 49(3):223–239 and 252–265, 1987.

[7] Jorma Rissanen. Fisher information and stochastic complexity. *IEEE Transactions on Information Theory*, 42(1):40–47, January 1996.

[8] Jorma Rissanen and G. G. Langdon, Jr. Universal modeling and coding. *IEEE Transactions on Information Theory*, 27:12–23, 1981.

[9] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[10] C. S. Wallace and D. M. Boulton. An information measure for classification. *Computer Journal*, 11(2):185–194, 1968.

[11] C. S. Wallace and P. R. Freeman. Estimation and inference by compact coding. *Journal of the Royal Statistical Society (Series B)*, 49(3):240–265, 1987.

[12] Richard S. Zemel. *A minimum description length framework for unsupervised learning*. PhD thesis, University of Toronto, Canada, 1993.

[13] Richard S. Zemel and Geoffrey E. Hinton. Developing population codes by minimizing description length. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *NIPS 6*, pages 11–18, San Francisco, 1994. Morgan Kaufmann.

ENSEMBLE LEARNING FOR INDEPENDENT COMPONENT ANALYSIS

Harri Lappalainen

In *Proceedings of the First International Workshop on Independent Component Analysis and Blind Signal Separation, ICA'99*, (Aussois, France, Jan. 11-15), pp. 7-12, 1999.

Abstract

In this paper, a recently developed Bayesian method called ensemble learning is applied to independent component analysis (ICA).

Ensemble learning is a computationally efficient approximation for exact Bayesian analysis. In general, the posterior probability density function (pdf) is a complex high dimensional function whose exact treatment is difficult. In ensemble learning, the posterior pdf is approximated by a more simple function and Kullback-Leibler information is used as the criterion for minimising the misfit between the actual posterior pdf and its parametric approximation. In this paper, the posterior pdf is approximated by a diagonal Gaussian pdf.

According to the ICA-model used in this paper, the measurements are generated by a linear mapping from mutually independent source signals whose distributions are mixtures of Gaussians. The measurements are also assumed to have additive Gaussian noise with diagonal covariance.

The model structure and all parameters of the distributions are estimated from the data.

1 Introduction

Recently there has been a lot of interest in Bayesian methods but few applications for unsupervised learning. One of the most important benefits of Bayesian methods is the possibility for model comparison. In supervised learning, cross validation or other methods can be used. For unsupervised learning this is usually not possible, however, since reconstruction error decreases also for the test set as the complexity of the model increases. The ability to optimise the structure of the model is thus particularly valuable in unsupervised learning.

Both ensemble learning, first used in [1], and independent component analysis using mixture of Gaussians model for sources, first used in [2], are existing techniques, but they have not been combined previously.

The reader is assumed to have basic knowledge about Bayesian probability theory and ICA.

2 Ensemble learning

Assume we would like to make a prediction, decision, etc., based on measurements and some kind of models. From the axioms of Bayesian probability theory it follows that all the models should be used in the process and the models should be weighted according

to the posterior probabilities of the models. This averaging over models is the essence of Bayesian analysis.

Usually the models include unknown real values and therefore the posterior probability is expressed by a posterior pdf. Unfortunately the posterior pdf is typically a complex high dimensional function whose exact treatment is difficult. In practice, it has to be approximated in one way or another.

In ensemble learning, a parametric computationally tractable approximation – an ensemble – is chosen for the posterior pdf. Let P denote the exact posterior pdf and Q the ensemble. The misfit between P and Q is measured with Kullback-Leibler information I_{KL} between Q and P.

$$I_{\mathrm{KL}}(Q;P) = E_Q\left\{\ln \frac{Q}{P}\right\}$$

The parameters of the ensemble are optimised to fit the posterior by minimising $I_{\mathrm{KL}}(Q;P)$.

Ensemble learning was first used in [1] where it was applied to a multi-layer perceptron with one hidden layer. Since then it has been used e.g. in [3-9].

2.1 Model selection

An important special case of approximation of the posterior pdf is the model selection. The posterior pdf is typically multimodal, but often almost all of the probability mass is located around the largest peak of the posterior pdf. When there is a lot of data compared to the complexity of the models, this is almost always the case. In our case, approximating the posterior pdf with only one peak is usually reasonably accurate.

Notice that the posterior density itself has no special meaning regarding the averaging over models; only the probability mass matters. A broad peak with low density can be more important than a sharp peak with high density. Over-learning results in high but very narrow peaks. The Kullback-Leibler information automatically takes into account the probability mass and is therefore robust against over-learning.

3 Model for the measurements

The measurements vectors $\{x(t)\}$ are assumed to be generated by a linear mapping A from mutually independent source signals $\{s(t)\}$ and additive Gaussian noise $\{v(t)\}$.

$$x(t) = As(t) + v(t)$$

The components $v_i(t)$ of the noise are assumed to have means b_i and variances $e^{2\sigma_i}$. Another way to put this is to say that $x(t)$ has Gaussian distribution with mean $As(t) + b$ and diagonal covariance with components $e^{2\sigma_i}$. Each component A_{ij} of the linear mapping is assumed to have zero mean and unit variance.

The distribution of each source signal is a mixture of Gaussians (MOG).

$$p(s_i(t)|c_i, S_i, \gamma_i) = \frac{\sum_j e^{c_{ij}} \mathcal{G}(s_i(t); S_{ij}, e^{2\gamma_{ij}})}{\sum_j e^{c_{ij}}}$$

The parameters c_{ij} are the logarithms of mixture coefficients, S the means and γ the logarithms of the standard deviations of the Gaussians[1] (here $\mathcal{G}(a; b, c)$ denotes a Gaussian distribution over a with mean b and variance c).

[1]The Gaussians are parametrised by the logarithms of the standard deviations in order to make the future assumption of roughly Gaussian posterior pdf valid.

The distributions of parameters c_{ij}, S_{ij}, γ_{ij}, b_i and σ_i are $\mathcal{G}(c_{ij}; 0, e^{2\alpha})$, $\mathcal{G}(S_{ij}; 0, e^{2\epsilon})$, $\mathcal{G}(\gamma_{ij}; \Gamma, e^{2\delta})$, $\mathcal{G}(b_i; B, e^{2\epsilon})$ and $\mathcal{G}(\sigma_i; \Sigma, e^{2\eta})$.

The prior distribution of the hyperparameters α, ϵ, Γ, δ, B, β, Σ and η is assumed to be uniform in the area of reasonable values for the hyperparameters.

To summarise: the eight hyperparameters are assigned flat prior pdfs. The distributions of other parameters are defined hierarchically from these using Gaussian distributions each parametrised by the mean and the logarithm of the standard deviation. The joint pdf of $\{x(t), s(t), A, b, \sigma, c, S, \gamma, \alpha, \epsilon, \Gamma, \delta, B, \beta, \Sigma, \eta\}$ is simply the product of the independent pdfs.

4 Diagonal Gaussian ensemble

Given the measurements, the unknown variables of the model are the source signals, the mixing matrix, the parameters of the noise and source distributions and the hyperparameters. The posterior P is thus a pdf of all these unknown variables. For notational simplicity, we shall sometimes denote these n variables by $\theta_1, \theta_2, \ldots, \theta_n$.

In order to make the approximation of the posterior pdf computationally tractable, we shall choose the ensemble Q to be a Gaussian pdf with diagonal covariance. The ensemble has twice as many parameters as there are unknown variables in the model because each dimension of the posterior pdf is parametrised by a mean and variance in the ensemble. A hat over a symbol denotes the mean and a tilde the variance of the corresponding variable.

$$Q(\theta_1, \ldots, \theta_n) = \prod_{i=1}^{n} \mathcal{G}(\theta_i; \hat{\theta}_i, \tilde{\theta}_i)$$

The factorised ensemble makes the computation of the Kullback-Leibler information $I_{\mathrm{KL}}(Q; P)$ simple since the logarithm can be split into a sum of terms: the terms $E_{\mathcal{G}(\theta_i)}\{\ln \mathcal{G}(\theta_i)\} = -1/2 \ln 2\pi e \tilde{\theta}_i$ (entropies of Gaussian distributions) and terms $-E_Q\{\ln P_i\}$, where P_i are the factors of the posterior pdf. Notice that the posterior pdf factorises into simple terms due to the hierarchical structure of the model; the posterior pdf equals to the joint pdf divided by a normalising term.

To see how to compute the terms $-E_Q\{\ln P_i\}$, let $P_i = p(\gamma_{ij}|\Gamma, \epsilon) = \mathcal{G}(\gamma_{ij}; \Gamma, \epsilon)$.

$$-E_Q\{\ln P_i\} = E_Q\{\frac{(\gamma_{ij} - \Gamma)^2 e^{-2\epsilon} + \ln 2\pi}{2} + \epsilon\} \tag{1}$$

It is easy to show that this equals to

$$\frac{[(\hat{\gamma}_{ij} - \hat{\Gamma})^2 + \tilde{\gamma}_{ij} + \tilde{\Gamma}]e^{2(\tilde{\epsilon} - \hat{\epsilon})} + \ln 2\pi}{2} + \hat{\epsilon},$$

since according to the choice of Q, the parameters γ_{ij}, Γ and ϵ have independent Gaussian distributions.

The most difficult terms are the expectations of $\ln p(s_i(t)|c_i, S_i, \gamma_i)$. Approximation for the expectation of this form is given in appendix A.

The normalising term in the posterior pdf only depends on those variables which are given, in this case $\{x(t)\}$, and can therefore be neglected when minimising the Kullback-Leibler information $I_{\mathrm{KL}}(Q; P)$.

5 Simulations

5.1 Data

Speech data was used for the simulations: 30 s of Finnish speech was digitised with 16 kHz sampling rate and high-pass filtered. Power spectra were computed every 8 ms using short time Fourier transformations with Hamming windows of length 16 ms. This results in 3749 vectors of dimension 128. Energy was computed for 34 channels whose spacing imitates the frequency scale of human ear. Logarithms were taken from the energies after adding small constants. The final data thus consisted of 3749 vectors of dimension 34.

This particular preprocessing was chosen because it is typical in current speech recognition systems.

5.2 Minimisation

The ensemble Q was fitted to the posterior pdf by minimising $I_{\mathrm{KL}}(Q; P)$. The particular technique for minimisation is not important regarding the results; everybody can use their favourite algorithm. In these simulations, a variation of Newton's method was used. The parameters $\hat{s}(t)$ and $\tilde{s}(t)$ were iterated 15 times for each measurement vector $x(t)$. Other parameters were updated after going through all the measurements, and this was iterated 200 times. The number of iterations was chosen conservatively. The details of the minimisation procedure will be given in [10].

5.3 Results

Several different structures for the model were tested. The number of source signals and the number of Gaussians in the mixtures was varied. The number of Gaussians in the mixtures was same for all sources in each network. This is not to say that it would not be perfectly simple to optimise the number of Gaussians for each source separately.

It turned out that the Kullback-Leibler information was minimised by a network with 23 dimensional source signals whose distributions were mixtures of three Gaussians. There were 87 292 unknown variables in the model: 86 227 in $s(t)$; 782 in A; 68 in b and σ; 207 in c, S and γ; and 8 in hyperparameters.

Figures 1–3 show the basis vectors, histograms and reconstructed distributions of the 23 source signals of the best network. The ordering in all three figures is the same. The basis vectors in figure 1 probably do not seem very interesting for someone who is not working with speech recognition. The basis is fairly close to cosine transformation which is widely used for processing the spectra in speech recognition.

The histograms in figure 2 and the corresponding distributions in figure 3 show that the algorithm works. The model has captures the salient features of the source distributions, some of which are multimodal, skewed or kurtotic.

The second best fit was obtained by a network with two Gaussian in the mixtures, but the probability mass it captured[2] was over 10^{76} times smaller! It is therefore reasonable to approximate the whole posterior pdf of all model structures and parameters by an ensemble with a peak in only one model structure. It is also evident that in this case, any prior information about the model structure has no significance.

[2]In [10] it will be explained how the relative probability masses of different models can be compared using the Kullback-Leibler information computed by the algorithm. See also [9].

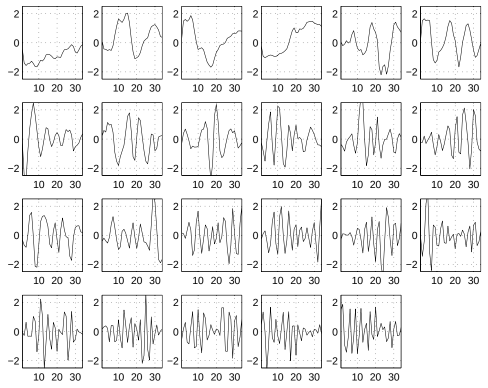

Figure 1: Each row vector of $\hat{A}$ is a 34 dimensional basis vector corresponding to one source. The frequency increaces from left to right in all the subimages.

For comparison, also models with only one Gaussian in the mixtures were tested. In this case the logarithmic mixture coefficients c_{ij} can be dropped out from the model. The best model with only one Gaussian was found to have over 10^{1238} times less probability mass. This shows that the algorithm agrees with human eye: it is clear that the distributions of at least some of the source signals are far from Gaussian.

6 Discussion

6.1 Benefits

Probably the most important benefit of Bayesian analysis regarding unsupervised learning is the ability to compare models. Not only can the different ICA-models be compared to each other, they can also be compared to vector quantisation or any other statistical models, provided that Bayesian analysis is also applied to these other models.

In ensemble learning, the treatment for missing values is very simple. Since they are unknown variables, they are included in the ensemble and therefore their distribution will be estimated similarly as for any other unknown variables.

6.2 Limitations

The method proposed here has limitations due to the simple structure of the ensemble. The posterior pdf of the unknown variables is often close to Gaussian, but there can be significant correlations. On the other hand, as the algorithm tries to fit the ensemble to the posterior pdf, it tries to find a peak in the posterior which would satisfy the diagonality assumption. In the simulations with only one Gaussian in the mixtures, for instance, the linear mapping found by the algorithm will orthogonal because that makes the sources independent in the posterior pdf.

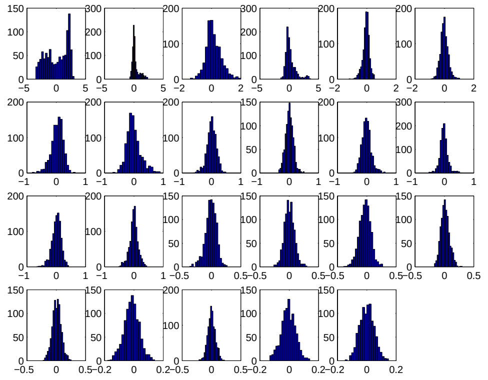

Figure 2: The histograms of the means $\hat{s}_i(t)$ of the sources.

There are two significant cases where the factorial assumption for the ensemble is too strong. If the row vectors of the linear mapping A are far from orthogonal, the source signals are correlated in the posterior pdf. Another case is when the amount of noise in the data is small and there are not very many data samples. In that case the components of the linear mapping are correlated with the source signals in the posterior pdf.

The limitations can be overcome by adding off-diagonal terms to the covariance matrix of the ensemble, but then the formulas for the Kullback-Leibler information become more complicated. Full covariance matrix is, of course, out of the question: the ensemble already had 174 584 parameters and with full covariance matrix the number would have been almost 7.62×10^9. It is more feasible to try to find model structures which make the off-diagonal terms in the covariance matrix of the unknown variables small.

6.3 Relation to previous work

Many current ICA-algorithms do not estimate the noise level from the data. The ability to estimate the noise is not due to the Bayesian analysis, however. It stems from the generative model used here. Similar models have been used for instance in [2, 11]. The treatment in the latter is very similar to ours: a factorial ensemble was fitted to the posterior pdf of the source signals by minimising their Kullback-Leibler information. However, the posterior did not include other parameters than the sources and only the maximum of the posterior pdf was used. As argued in section 2.1, large density does not necessarily imply large mass, and therefore the EM-algorithm used in [11] does not necessarily find a probable model for the data, and can not, in any case, be used for comparing models with different structures.

6.4 Some generalisations

The most obvious generalisation would be to take into account the significant temporal correlations. Also, the linear mapping can be replaced by a nonlinear one. An interesting generalisation is to let the variances of the sources vary in time. According to this model,

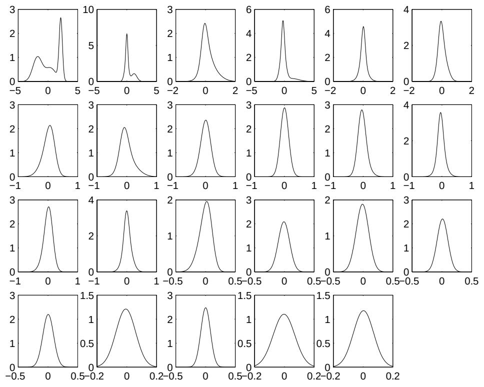

Figure 3: The distributions of the sources reconstructed from $\hat{c}$, $\hat{S}$ and $\hat{\gamma}$.

higher lever source signals modulate the variances of source signals.

Simulations with these and other variants will be published in [10]. All these models have significantly larger probabilities than the simple ICA-models studied in this paper.

References

[1] Geoffrey Hinton and Drew van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the COLT'93*, pages 5–13, Santa Cruz, California, 1993.

[2] Éric Moulines, Jean-François Cardoso, and Elisabeth Gassiat. Maximum likelihood for blind separation and deconvolution of noisy signals using mixture models. In *Proceedings of the ICASSP'97*, pages 3617–3620, Munich, Germany, 1997.

[3] David J. C. MacKay. Developments in probabilistic modelling with neural networks—ensemble learning. In *Neural Networks: Artificial Intelligence and Industrial Applications. Proceedings of the 3rd Annual Symposium on Neural Networks, Nijmegen, Netherlands, 14-15 September 1995*, pages 191–198, Berlin, 1995. Springer.

[4] Lawrence K. Saul, Tommi Jaakkola, and Michael I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

[5] David J. C. MacKay. Comparison of approximate methods for handling hyperparameters. *Neural Computation*. Submitted.

[6] David J. C. MacKay. Ensemble learning for hidden Markov models. Available from `http://wol.ra.phy.cam.ac.uk/`, 1997.

[7] David Barber and Christopher M. Bishop. Ensemble learning for multi-layer networks. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 395–401. MIT Press, 1998.

[8] David Barber and Bernhard Schottky. Radial basis functions: a bayesian treatement. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 402–408. MIT Press, 1998.

[9] Christopher M. Bishop, Neil Lawrence, Tommi Jaakkola, and Michael I. Jordan. Approximating posterior distributions in belief networks using mixtures. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 416–422. MIT Press, 1998.

[10] Harri Lappalainen. Ensemble learning for unsupervised neural networks. Technical report, Helsinki University of Technology, Laboratory of Computer and Information Science, 1998. In preparation.

[11] Hagai Attias. Independent factor analysis. *Neural Computation*, 1998. Submitted.

A Derivations

An approximation for $-E_Q\{\ln p(s_i(t)|c_i, S_i, \gamma_i)\}$ is derived here. The derivation makes use of the Jensen's inequality and second order Taylor's series expansion.

Let $g_{ij} = -\ln \mathcal{G}(s_i(t); S_{ij}, \gamma_{ij})$. The expectation to be approximated is then

$$E_Q\left\{\ln \sum_j e^{c_{ij}}\right\} - E_Q\left\{\ln \sum_j e^{c_{ij}-g_{ij}}\right\}.$$

Let us first concider the latter term. The logarithm of the sum is a strictly convex function of g_{ij}. By Jensen's inequality, moving the expectation inside a convex function cannot result in increment. Replacing the latter expectation by

$$E_{s_i(t),c_i}\left\{\ln \sum_j e^{c_{ij}-E_{S_{ij},\gamma_{ij}}\{g_{ij}\}}\right\}$$

can therefore only result in increment in the approximation. This is safe because we are trying to minimise the Kullback-Leibler information and approximating it above is thus conservative.

The expectation $E_{S_{ij},\gamma_{ij}}\{g_{ij}\}$ is similar to equation 1 and equals to

$$\frac{[(s_i(t) - \hat{S}_{ij})^2 + \tilde{S}_{ij}]e^{2(\tilde{\gamma}_{ij}-\hat{\gamma}_{ij})} + \ln 2\pi}{2} + \hat{\gamma}_{ij}.$$

Let us denote this by $\hat{g}_{ij}(s_i(t))$. At this point, the approximation equals to

$$E_{c_i}\left\{\ln \sum_j e^{c_{ij}}\right\} - E_{s_i(t),c_i}\left\{\ln \sum_j e^{c_{ij}-\hat{g}_{ij}(s_i(t))}\right\}.$$

The terms inside the expectations are functions of $s_i(t)$ and c_i.

Next, let us concider a second order Taylor's series expansion about $\hat{s}_i(t)$ and $\hat{c}_i$. Notice that the first order terms and all second order crossterms disappear in the expectations and only the constant and the pure second order terms remain. This is because the variables are independent in the ensemble.

The constant term will be

$$\ln \sum_j e^{\hat{c}_{ij}} - \ln \sum_j e^{\hat{c}_{ij} - \hat{g}_{ij}(\hat{s}_i(t))} \tag{2}$$

and the remaining second order terms of c_{ij}

$$E_{c_i}\left\{ \sum_j \frac{(c_{ij} - \hat{c}_{ij})^2}{2} \zeta_{ij}(1 - \zeta_{ij}) \right\}$$

and

$$-E_{c_i}\left\{ \sum_j \frac{(c_{ij} - \hat{c}_{ij})^2}{2} \xi_{ij}(1 - \xi_{ij}) \right\},$$

where

$$\zeta_{ij} = \frac{e^{\hat{c}_{ij}}}{\sum_k e^{\hat{c}_{ik}}}$$

and

$$\xi_{ij} = \frac{e^{\hat{c}_{ij} - \hat{g}_{ij}(\hat{s}_i(t))}}{\sum_k e^{\hat{c}_{ik} - \hat{g}_{ik}(\hat{s}_i(t))}}.$$

Taking the expectations yields

$$\sum_j \frac{\tilde{c}_{ij}}{2} [\zeta_{ij}(1 - \zeta_{ij}) - \xi_{ij}(1 - \xi_{ij})]. \tag{3}$$

The second order term of $s_i(t)$ will be

$$E_{s_i(t)}\left\{ \frac{(s_i(t) - \hat{s}_i(t))^2}{2} (\phi_i + \chi_i^2 - \psi_i) \right\},$$

where

$$\phi_i = \sum_j \xi_{ij} e^{2(\tilde{\gamma}_{ij} - \hat{\gamma}_{ij})},$$

$$\chi_i = \sum_j \xi_{ij} (\hat{s}_i(t) - \hat{S}_{ij}) e^{2(\tilde{\gamma}_{ij} - \hat{\gamma}_{ij})}$$

and

$$\psi_i = \sum_j \xi_{ij} \left[(\hat{s}_i(t) - \hat{S}_{ij}) e^{2(\tilde{\gamma}_{ij} - \hat{\gamma}_{ij})} \right]^2.$$

Taking the expectation yields

$$\frac{\tilde{s}_i(t)}{2} (\phi_i + \chi_i^2 - \psi_i). \tag{4}$$

At this point, the approximation of the original expectation is thus the sum of terms in equations 2–4.

Some care has to be taken with the approximation resulting from the Taylor's series expansion because it utilises only local information about the shape of the posterior pdf. For instance, if the mean $\hat{s}_i(t)$ happens to be in a valley between two Gaussians, the term in equation 4 will be negative. It then looks like the Kullback-Leibler information can be decreased by increasing $\tilde{s}_i(t)$. This only holds for small $\tilde{s}_i(t)$, however. At some point after the distribution of $s_i(t)$ has become broader than the separation between the two Gaussians, the Kullback-Leibler information starts to increase as $\tilde{s}_i(t)$ increases.

In order to avoid the problem, only positive terms of equations 3 and 4 will be included. The final approximation is thus equation 2 plus the positive terms of equations 3 and 4.

MULTI-LAYER PERCEPTRONS AS NONLINEAR GENERATIVE MODELS FOR UNSUPERVISED LEARNING: A BAYESIAN TREATMENT

Harri Lappalainen and Xavier Giannakopoulos

In *Proceedings of the Ninth International Conference on Artificial Neural Networks, ICANN'99*, (Edinburgh, UK, Sep. 7–10), pp. 19-24, 1999.

Abstract

In this paper, multi-layer perceptrons are used as nonlinear generative models. The problem of indeterminacy of the models is resolved using a recently developed Bayesian method called ensemble learning. Using a Bayesian approach, models can be compared according to their probabilities. In simulations with artificial data, the network is able to find the underlying causes of the observations despite the strong nonlinearities of the data.

1 Introduction

Many types of unsupervised learning can be viewed as generative learning where the goal is to find a model which explains how the observations were generated. The hypothesis is that there are latent variables which have generated the observations by an unknown mapping. The goal of the learning is to identify the latent variables and the unknown mapping.

The success of the model depends on how well it can capture the structure of the phenomena underlying the observations. Sometimes the process is well characterised by assuming a discrete latent variable which produces different observations at different states. Then the generative model used in vector quantisation is appropriate. If there is reason to assume that several independent latent variables have generated the observations via a linear mapping, then the model used in independent component analysis suits the problem well.

In many realistic cases it is reasonable to assume that there are several latent variables which affect the observations nonlinearly. One example could be the effect of pressure and temperature on the properties of the end product of a chemical process. Although many effects in real world are locally linear, the overall effects are almost always nonlinear. Also, there are usually several factors whose nature and effect on the observations are completely unknown and whose direct measurement is impossible for practical reasons.

The goal of this work is to develop methods for inferring the hidden causes, the latent variables, from the observations alone. The nonlinear mapping from the unknown latent variables to the observations is modelled with the familiar multi-layer perceptron network (MLP).

2 Bayesian learning

Given the observed data, there are usually more than one way to explain it. With a flexible model family — like MLP-networks — there is always an infinite amount of explanations

and it could be difficult to choose among them. Choosing too complex a model would result in overlearning, a situation where one not only finds the underlying causes of the observations but also makes up meaningless explanations for the noise always present in real signals. Choosing too simple a model results in underlearning, i.e., would leave some of the true causes hidden.

The solution to the problem is that no single model should, in fact, be chosen. Probability theory tells that all the explanations should be taken into account and weighted according to their posterior probabilities. This approach, known as Bayesian learning, optimally solves the tradeoff between under- and overlearning.

The posterior probability densities of too simple models are low because they leave much of the data unexplained while the peaks of the posterior probability density function (pdf) of too complex models are high but also very narrow. This is because a complex model is very sensitive to changes in its parameters. Due to the narrow peaks, too complex models occupy little probability mass and therefore contribute little to expectations weighted by the probabilities.

2.1 Parametric approximation of the posterior pdf

In practice, exact treatment of the posterior pdf of the models is impossible and the posterior pdf needs to be approximated. The existing methods for doing this can be roughly divided into stochastic sampling and parametric approximation. Stochastic sampling typically yields better approximations but is also computationally much more expensive. We therefore opt for the computationally efficient parametric approximation which usually yields satisfactory results.

A standard approach for parametric approximation is the Laplace's method. One variation was introduced to the neural networks community by MacKay, who called his method the evidence framework: one first finds a (local) maximum point of the posterior pdf and then applies a second order Taylor's series approximation for the logarithm of the posterior pdf. This amounts to applying the Gaussian approximation to the posterior pdf.

Unfortunately, also Laplace's method can suffer from overlearning. Recall that too complex models have very high posterior probability densities. Therefore finding the maximum point of the posterior pdf focuses the search on too complex models. In the end, the second order Taylor's series approximation will reveal that the peak is narrow, but then it is already too late.

2.2 Ensemble learning

Ensemble learning [3, 6], also known as variational learning, is a recently developed method for parametric approximation of posterior pdfs where the search takes into account the probability *mass* of the models. Therefore, it does not suffer from overlearning. The basic idea is to minimise the misfit between the posterior pdf and its parametric approximation.

Let P denote the exact posterior pdf and Q its parametric approximation. The misfit is measured with the Kullback-Leibler divergence between P and Q and thus the cost function C_{KL} is

$$C_{\mathrm{KL}} = E_Q \left\{ \log \frac{Q}{P} \right\}. \tag{1}$$

Notice that the Kullback-Leibler divergence involves an expectation over a distribution and, consequently, is sensitive to probability mass rather than probability density.

3 Model structure

The basic model structure is the ordinary MLP network. It has a slight improvement, however, and to motivate this, let us first consider how the learning with generative models looks like in practice.

As usually, we start with a random initialisation of the weights. Then we take the first observed data vector and find those values of the latent variables which best explain the observed data. In vector quantisation, for instance, this is very easy. The latent variable is the index of the model vector which is used for representing the data, and therefore the best value for the latent variable is simply the index of the closest model vector.

Inverting an MLP network is harder, however, and we are going to use gradient descent for doing it. Usually back propagation is used for updating the weights, but it can be used for adapting the unknown inputs, the latent variables as well. Typically the gradient descent has to be iterated several times in order to find the optimal latent variables.

To put the same thing differently, in supervised learning we are presented with the inputs and desired outputs. In our case the latent variables play the role of inputs and the observations play the role of outputs. The difference is that the latent variables are unknown and to find them, the model has to be inverted.

Once the optimal latent variables are found, the inputs of the MLP network are known and the learning proceeds as in supervised learning: the weights are adapted so as to make the mapping from the found latent variables to the observed data even better. Again we can draw a parallel from vector quantisation: in most learning algorithms the best matching model vector is moved even closer to the input.

Then we take the next data vector, find the latent variables that best describe the data by iterating the gradient descent a few times, adapt the weights, and so on.

In the beginning the learning can be slow, however. When the model is random, no values of the latent variables are able to explain much of the data. The optimal latent variables can be more or less random, and without sensible inputs it is difficult to adapt the mapping. It would therefore seem that learning is bound to be slower with unsupervised generative models than with supervised models.

Luckily, it turns out that the situation is much better because the mapping can be learned layer by layer starting from the layers closest to the observations. The point is to create parts of the network only when they may have meaningful input. In this process the model is refined when getting closer to the solution.

In the beginning, only the linear layer is created, together with first layer latent variables which act as training wheels for the network. At this point the mapping is linear and the network quickly finds some meaningful values for the first layer weights.

After the first layer has found a rough representation, the second, nonlinear layer is added on top of the first layer. Since the first layer weights already have reasonable values, the second layer learns much faster. Initially the data is represented mainly by the first layer latent variables, but gradually the second layer latent variables take over and the first layer latent variables become silent.

We can now formalise the model used in this work. Let $\mathbf{x}(t)$ denote the observed data vector at time t; $\mathbf{s}_1(t)$ and $\mathbf{s}_2(t)$ the vectors of latent variables of the first and the second layer at time t; $\mathbf{A}$ and $\mathbf{B}$ the matrices containing the weights on the first and the second layer, respectively; $\mathbf{b}$ the vector of biases for the second layer and $\mathbf{f}$ the vector of nonlinear activation functions. As all real signals contain noise, we shall assume that observations are corrupted by Gaussian noise denoted by $\mathbf{n}(t)$. Using this notation, the model for the

observations passes through the three phases described below:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{A}\mathbf{s}_1(t) + \mathbf{n}(t) && (2)\\ \mathbf{x}(t) &= \mathbf{A}\left[\mathbf{f}\left(\mathbf{B}\mathbf{s}_2(t) + \mathbf{b}\right) + \mathbf{s}_1(t)\right] \\ &\quad + \mathbf{n}(t) && (3)\\ \mathbf{x}(t) &= \mathbf{A}\left[\mathbf{f}\left(\mathbf{B}\mathbf{s}_2(t) + \mathbf{b}\right)\right] + \mathbf{n}(t). && (4) \end{aligned}$$

The nonlinearity most often used in MLP networks is the hyperbolic tangent, but it has a disadvantage from the point of view of this application. It saturates for large values of its inputs which can cause problems during the inversion of the model. For the sake of numerical stability, the inverse hyperbolic sine, $\sinh^{-1}$, is chosen instead. It is also a sigmoidal function but instead of saturating it behaves logarithmically for large values.

The latent variables are assumed to be independent and Gaussian. The independence assumption is natural as the goal of the model is to find the underlying independent causes of the observations. If the latent variables were dependent, then they would presumably have a common cause which should be modelled by yet another latent variable.

Even the Gaussianity assumption is usually not unrealistic. The network has nonlinearities which can transform the Gaussian distributions to virtually any other regular distribution. This is why with linear models it makes a difference whether the latent variables are assumed to have Gaussian, as in PCA, or non-Gaussian distributions, as in ICA, but for nonlinear models these assumptions do not make such a great difference. It may, of course, sometimes be that an explicit model of a non-Gaussian distribution, e.g., by mixtures of Gaussians as in [6], is simpler than an implicit model with nonlinearities.

The parameters of the network are: (1) the weight matrices $\mathbf{A}$ and $\mathbf{B}$ and the vector of biases $\mathbf{b}$; (2) the parameters of the distributions of the noise, latent variables and column vectors of the weight matrices; and (3), hyperparameters which are used for defining the distributions of the biases and the parameters in the group (2). For simplicity, all the parametrised distributions are assumed to be Gaussian.

This kind of hierarchical description of the distributions of the parameters in the model is a standard procedure in probabilistic modelling. Its strength is that knowledge about equivalent status of different parameters can be easily incorporated. All the variances of the noise components, for instance, have a similar status in the model and this is reflected by the fact that their distributions are assumed to be governed by common hyperparameters. Often there is some vague prior information about the distributions of the hyperparameters, but the amount of information is, in any case, very small compared to the amount of information in the data. Here the hyperparameters are assigned flat priors.

4 Cost function

The cost function was already outlined in section 2.2. We can now go into more detail. Let us denote $X = \{\mathbf{x}(t)|t\}$ and $S = \{\mathbf{s}_1(t), \mathbf{s}_2(t)|t\}$ and let $\boldsymbol{\theta}$ denote all the unknown parameters of the model. For notational simplicity, let us denote all the unknown variables by $\boldsymbol{\xi} = \{S, \boldsymbol{\theta}\}$. The cost function is then

$$C_{\mathrm{KL}} = \int d\boldsymbol{\xi} Q(\boldsymbol{\xi}) \log \frac{Q(\boldsymbol{\xi})}{P(\boldsymbol{\xi}|X)}. \qquad (5)$$

The two things needed for equation 5 are the exact formulation of the posterior density $P(\boldsymbol{\xi}|X)$ and its parametric approximation $Q(\boldsymbol{\xi})$.

According to the Bayes' rule, the posterior pdf of the unknown variables S and $\boldsymbol{\theta}$ is

$$P(S, \boldsymbol{\theta}|X) = \frac{P(X|S, \boldsymbol{\theta})P(S|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(X)}.$$

The term $P(X|S, \boldsymbol{\theta})$ is obtained from equations 2–4; the distribution of the data is the same as for the noise $\mathbf{n}(t)$ except that its mean is corrected by the prediction given by the model. Let us denote the vector of the means of $\mathbf{n}(t)$ by $\boldsymbol{\mu}$ and the vector of the variances by $\boldsymbol{\sigma^2}$. The distribution $P(x_i(t)|\mathbf{s}_1(t), \mathbf{s}_2(t), \boldsymbol{\theta})$ is thus Gaussian with mean $\mathbf{a}_{i\cdot}[\mathbf{f}(\mathbf{B}\mathbf{s}_2 + \mathbf{b}) + \mathbf{s}_1] + \mu_i$ and variance σ_i^2. Here $\mathbf{a}_{i\cdot}$ denotes the ith row vector of $\mathbf{A}$. As usually, the noise components $n_i(t)$ are assumed to be independent and therefore $P(X|S, \boldsymbol{\theta}) = \prod_{t,i} P(x_i(t)|\mathbf{s}_1(t), \mathbf{s}_2(t), \boldsymbol{\theta})$.

The terms $P(S|\boldsymbol{\theta})$ and $P(\boldsymbol{\theta})$ are also products of simple Gaussian distributions and they are obtained directly from the definition of the model structure. The term $P(X)$ is not a function of any of the parameters of the model and can be neglected.

The approximation $Q(S, \boldsymbol{\theta})$ needs to be simple for mathematical tractability and computational efficiency. We assume that it is Gaussian density with a diagonal covariance matrix. This means that the approximation is a product of the independent distributions: $Q(\boldsymbol{\xi}) = \prod_i Q_i(\xi_i)$. The parameters of each $Q_i(\xi_i)$ are the mean and variance which will be denoted by $\hat{\xi}_i$ and $\tilde{\xi}_i$, respectively.

Both the posterior density $P(S, \boldsymbol{\theta}|X)$ and its approximation $Q(S, \boldsymbol{\theta})$ are products of simple Gaussian terms, which simplifies the cost function considerably: it splits into expectations of many simple terms. The terms of the form $E_Q\{\log Q_i(\xi_i)\}$ are the negative entropies for Gaussians and have the values $-(1 + \log 2\pi\tilde{\xi}_i)/2$. The most difficult terms are of the form $-E_Q\{\log P(x_i(t)|\mathbf{s}_1(t), \mathbf{s}_2(t), \boldsymbol{\theta})\}$. They are approximated by applying the second order Taylor's series expansions of the nonlinear activation functions as explained in [5]. The rest of the terms are expectations of simple Gaussian terms, whose expectations can be computed as in [6].

The cost function C_{KL} is a function of $\hat{\xi}_i$ and $\tilde{\xi}_i$, i.e., the posterior means and variances of the latent variables and the parameters of the network. This is because instead of finding a point estimate, a whole distribution will be estimated for the latent variables and the parameters during learning.

5 Simulations

5.1 Data

Artificial data was generated with a randomly initialised MLP network having one hidden layer with tanh-nonlinearities and a linear output layer. The network had a 2-10-5 structure and the data thus consisted of a two-dimensional manifold nonlinearly wrapped in five dimensions. The inputs had Gaussian distributions with unit covariance matrix. The five-dimensional outputs of the random network were further linearly embedded in a 10-dimensional space and finally, Gaussian noise was added to the data which consisted of 1000 vectors.

5.2 Learning scheme

Learning was done in batches which had two nested loops. The outer loop went through all the data vectors. For each data vector, the posterior distribution Q of the latent variables was adapted in the inner loop of length 15. The distributions of the rest of the model

parameters were updated at the end of each batch. The whole learning consisted of 200 batches.

Only the first, linear layer was generated in the beginning. The second, nonlinear layer was generated after 20 batches when the first layer had already found a rough representation for the data. To encourage the growth of the second layer, the standard deviations of the latent variables of the first layer were reduced by a factor of three after each batch during 10 batches starting from the creation of the second layer. If this phase is left out, the network easily gets stuck in a local minimum where the training wheels, the latent variables of the first layer, represent the data while the second layer remains silent.

The posterior variance $\tilde{\theta}_i$ of a parameter contains information about how certain the network is about the value of the parameter. This gives the network the very interesting property of being able to effectively prune away useless weights by increasing their posterior variances and thus decreasing the complexity of the network.

Some care has to be taken in the beginning of the learning since the network might get stuck in some unwanted local minima. When the weights of the network are random and the network is not able to find any structure in the data, the weights can get prematurely pruned away. To prevent this, the posterior variances of the weights and the latent variables were bounded for the first 50 batches.

5.3 Results

Several different structures were tested for the network and the best one was chosen according to the probability mass it occupied in the posterior pdf of all models. The most probable model had four latent variables and ten hidden units. Despite the difficulty of the problem, the network was able to discover the underlying causes of the observations. Out of the four latent variables, two corresponded to the original inputs that had generated the data while the other two had much smaller variances and were used by the network to represent the slight discrepancies between the original and the estimated nonlinear generative models. Notice that the two models used different nonlinearities and therefore can never be exactly the same.

Figure 1 shows the scatter plots of the four estimated latent variables (x-axes) versus the two original inputs (y-axes). The first original input (upper row) correlated with the second latent variable while the second original input correlated with the fourth latent variable. The figure shows that the first and third latent variables had much smaller variance than the two others.

For comparison, four independent components extracted by linear ICA [1] are shown in similar scatter plots in figure 2. As the data is severely nonlinear, linear ICA performs very poorly. None of the retrieved sources correspond to the original inputs and all four sources are used for representing the data. These results show that the nonlinearity was quite strong and, consequently, the problem of finding the underlying latent variables very difficult.

6 Discussion

The combination of MLP networks, generative learning and full Bayesian analysis is novel — to the best of our knowledge — although the individual parts have been published

[1]The FastICA MATLAB-toolbox, available at `http://www.cis.hut.fi/projects/ica/fastica/`.

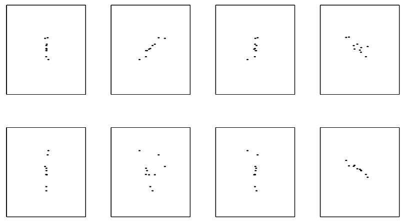

Figure 1: Scatter plots of the two original inputs and the posterior means of the four estimated latent variables.

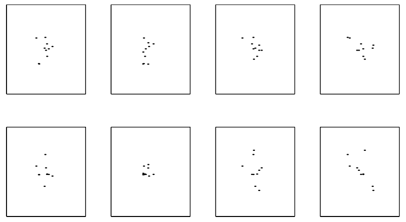

Figure 2: Scatter plots of the original inputs and the four sources estimated by linear ICA.

earlier. For instance, MLP networks were used as generative models in [7]. The model was inverted by gradient descent as in this work, but Bayesian analysis was not applied.

Rectified Gaussian belief networks were used as generative models in [2], but the Bayesian analysis was restricted to the posterior distributions of the latent variables and stochastic sampling was used instead of parametric approximation. Also [1] neglects the Bayesian treatment of the parameters of the network. With flexible models having a large number of parameters, it is important to take into account also the complexity of the nonlinear mapping. Restricting the Bayesian approach to the latent variables can lead to problems with overlearning.

Methods based on minimising the description length of the model are closely related, often equivalent, to Bayesian learning [3, 4, 6] since the description length is by definition the minus logarithm of the probability mass of the model. The description length of an auto-associative MLP model was minimised in [4]. The disadvantage of auto-associative models is that they need to learn both the generative mapping and its inversion and the learning can thus be slow.

7 Conclusion

Using MLP networks as generative models was proven feasible in simulations with artificial data. The network was able to retrieve the original inputs which had generated the data. The difficulty of the problem is apparent from the results obtained with the linear ICA-model. Bayesian learning was used for solving the indeterminacy of the unknown mapping.

The Bayesian approach is particularly valuable for unsupervised learning due to its robustness against overlearning and the ability to compare models. Other techniques, such as cross-validation, are available for supervised learning but they are not applicable for unsupervised learning.

The Bayesian approach was implemented using ensemble learning, which is an efficient method for approximating the posterior distributions. Its main advantage over the traditional Laplace's method is that the search for good models is focused on those areas of the model space which occupy large probability mass, as opposed to searching for large probability densities.

References

[1] Hagai Attias. Hierarchical ICA belief networks. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*. MIT Press, 1999. In press.

[2] Zoubin Ghahramani and Geoffrey 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. MIT Press, 1998.

[3] Geoffrey E. Hinton and Drew van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the COLT'93*, pages 5–13, Santa Cruz, California, 1993.

[4] Sepp Hochreiter and Jürgen Schmidhuber. LOCOCODE performs nonlinear ICA without knowing the number of sources. In *Proceedings of the ICA'99*, pages 149–154, Aussois, France, 1999.

[5] Harri Lappalainen. Using an MDL-based cost function with neural networks. In *Proceedings of the IJCNN'98*, pages 2384–2389, Anchorage, Alaska, 1998.

[6] Harri Lappalainen. Ensemble learning for independent component analysis. In *Proceedings of the ICA'99*, pages 7–12, Aussois, France, 1999.

[7] Jong-Hoon Oh and H. Sebastian Seung. Learning geneartive models with the up-propagation algorithm. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 605–611. MIT Press, 1998.

ENSEMBLE LEARNING

Harri Lappalainen and James W. Miskin

In *Advances in Independent Component Analysis* (M. Girolami, ed.), pp. 76-92, Springer-Verlag, 2000.

Abstract

This chapter gives a tutorial introduction to Ensemble Learning, a recently developed Bayesian method. For many problems it is intractable to perform inferences using the true posterior density over the unknown variables. Ensemble Learning allows the true posterior to be approximated by a simpler approximate distribution for which the required inferences are tractable.

1 Introduction

When we say we are making a model of a system, we are setting up a tool which can be used to make inferences, predictions and decisions. Each model can be seen as a hypothesis, or explanation, which makes assertions about the quantities which are directly observable and which can only be inferred from their effect on observable quantities.

In the Bayesian framework, knowledge is contained in the conditional probability distributions of the models. We can use Bayes' theorem to evaluate the conditional probability distributions for the unknown parameters, y, given the set of observed quantities, x, using

$$p(y|x) = \frac{p(x|y)\,p(y)}{p(x)} \tag{1}$$

The prior distribution $p(y)$ contains our knowledge of the unknown variables before we make any observations. The posterior distribution $p(y|x)$ contains our knowledge of the system after we have made our observations. The likelihood, $p(x|y)$, is the probability that the observed data will be observed given a specific set of values for the unknown parameters.

There is no clear cut difference between the prior and posterior distributions, since after a set of observations the posterior distribution becomes the prior for another set of observations.

In order to make inferences based on our knowledge of the system, we need to marginalise our posterior distribution with respect to the unknown parameters of the model. For instance in order to obtain the average values of the unknown parameters we would need to perform the expectation

$$\bar{y} = \int y p(y|x)\, dy \tag{2}$$

Alternatively we may be trying to use our model of the system to make decisions about which action to take. In this case we would like to choose the action which maximises some utility function. In this case the expected utility is found by marginalising the utility function over the posterior density of the models. An example would be hypothesis testing

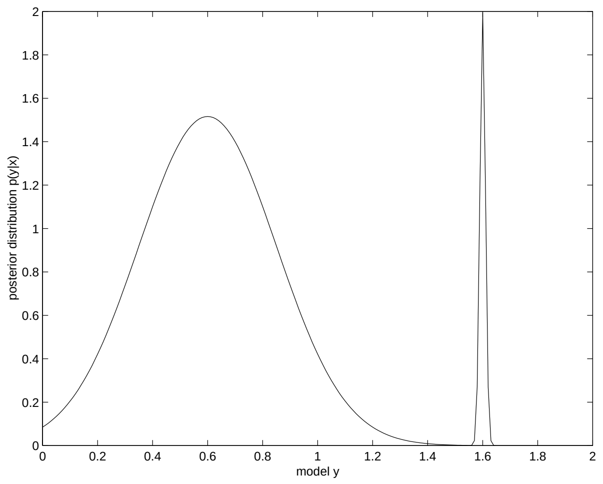

Figure 1: Schematic illustration of overfitting to the data. If the true posterior over the models, y, is of this form then choosing the model that maximises the posterior probability will mean choosing the model in the sharp peak. But near the maximum the posterior density is sharp and so highly dependent on the model parameters. Therefore the model will explain the observed data very well, but may not generalise to further observations.

where we have a number of explanations for the cause of a set of observed data. By having a different model for each hypothesis, we could choose the model that maximises the expected utility.

In this chapter we shall motivate the idea of considering the posterior distribution to be the result of any experiment instead of just considering a point in the model space. Section 3 will discuss different methods of approximating the posterior when it is intractable to make inferences based on the true posterior. Section 4 will introduce the idea of using Ensemble Learning to approximate the true posterior by a simpler separable distribution. Section 5 will discuss the construction of probabilistic models, both in supervised and unsupervised learning. Section 6 will give examples of using ensemble learning in both a fixed form and free form approximation.

2 Posterior averages in action

Probability theory tells us that the optimal generalisation is the one resulting from a Bayesian approach. Overfitting to the data means that we are making conclusions that the data does not support. Alternatively underfitting means that our conclusions are too diffuse.

Overfitting is an artifact resulting from choosing only one explanation (model) for the observations. Figure 1 shows a hypothetical posterior distribution. If the model is chosen

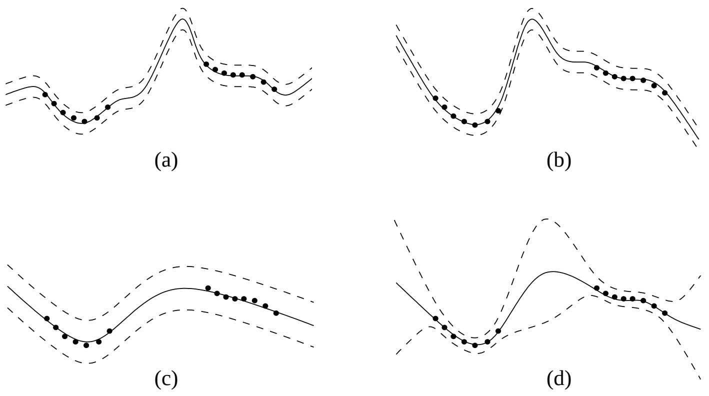

Figure 2: A schematic illustration of averaging over posterior of models. The data points are denoted by circles and the black lines show the MLP fit. The dashed lines show the error bars the model gives. The models in plots a and b assume fairly small noise. The model in plot c assumes larger noise. The average of the models over the posterior pdf of the models gives larger error bars in the areas where there are no observations.

to maximise the posterior probability then the model will be chosen to be in the narrow peak. The problem is that the peak only contains a fraction of the total probability mass. This means that the model will explain the observations very well, but will be very sensitive to the values of the parameters and so may not explain further observations. When making predictions and decisions it is the position of the probability mass that counts and not the position of a maximum.

In order to solve these problems it is necessary to average over all possible hypotheses. That is we should average over all possible models, but weight them by the posterior distribution that we obtain from our observations.

It is important to note that averaging does not mean computing the average of parameter values and then using that as the best guess. For instance in digit recognition the posterior distribution may have a peak in the model for “1”s and in the model for “9”s. Averaging over the posterior does not mean that you compute the average digit as a “5” and then use that as the best guess, it means that you should prepare for the possibility that the digit can be either “1” or “9”.

Figure 2 shows a schematic example of fitting a Multi-Layer Perceptron (MLP) to a set of data. The data is fairly smooth and has only a little noise, but there is a region of missing data. The best regularised MLP is able to fit the data quite well in those areas where there are data points and so the noise level will be estimated to be low. Therefore the MLP will give tight error bars even in the region where there is no data to support the fit. This is overfitting because the conclusion are more specific than the data supports. If we were to later obtain some data in the missing region, it is plausible that it would lie significantly outside the MLP fit and so the posterior probability of the chosen model could rapidly drop as more data is obtained.

Instead of chosing a specific MLP it is better to average over several MLPs. In this

case we would find that there are multiple explanations for the missing data. Therefore the average fit will give tight error bars where there is data and broad error bars in the regions where there is no data.

If the estimate for the level of noise in the data were estimated to be too high then the error bars for the areas with missing data might be good but the error bars for the areas with data would too broad and we would suffer from underfitting. Alternatively if the noise level is estimated to be too low, the error bars will be too narrow where there is data. For intermediate noise levels we could suffer from overfitting in some regions and underfitting in others.

In order to fix this we could allow the noise level to be part of our model and so the noise level could vary. The problems with overfitting would now be changed to overfitting the noise model. The solution is to average the noise under the posterior distribution which would include the MLP parameters and the noise.

3 Approximations of posterior pdf

As we have shown earlier, the posterior distribution is a synonym for our available knowledge of a system after we have made a set of observations. In order to use this knowledge we will typically be required to marginalise the posterior or to evaluate expectations of functions under the posterior distribution. In many problems it is intractable to perform the necessary integrals.

If we look again at Bayes equation we have

$$p\left(y\,|x\right) = \frac{p\left(x\,|y\right)p\left(y\right)}{p\left(x\right)} \tag{3}$$

It is easy to compute one point in the joint density $p\left(x,y\right)$ (the numerator of the posterior distribution) but in general evaluating the denominator, $p\left(x\right)$, is difficult. Similarly marginalising the posterior distribution is difficult.

Therefore it is necessary to approximate the posterior density by a more tractable form for which it is possible to perform any necessary integrals. We cannot take a point estimate (such as the MAP estimate) because this leads to overfitting as shown earlier. This is because the MAP estimate does not guarantee a high probability mass in the peak of the posterior distribution and so the posterior distribution may be sharp around the MAP estimate. We would like to approximate the probability mass of the posterior.

There are in general two types of approximation that retain the probability mass of the true posterior distribution. The first type is the stochastic approximation and the second type is the parametric approximation. In a stochastic approximation (such as Markov-chain Monte Carlo method) the aim is to perform the integrations by drawing samples from the true posterior distribution, [3]. The average of any function is then found by finding the average value of the function given all of the samples from the posterior.

In a parametric approximation (such as Laplace approximation, [6]) the posterior distribution is approximated by an alternative function (such as a Gaussian) such that it is much simpler to perform any necessary approximations.

The problem with the stochastic methods is that when performing a stochastic approximation it is necessary to wait until the sampler has sampled from all of the mass of the posterior distribution. Therefore testing for convergence can be a problem. The problem with the parametric approach is that the integrals that are being performed are not exactly the same as those that would be performed when the true posterior is used. Therefore while the stochastic approximation has to give the correct answer (eventually) the parametric approxmation will give an approximate answer soon (it is the 'quick and dirty' method).

Model selection can be seen as a special form of approximating the posterior distribution. The posterior distribution could contain many peaks, but when there is lots of data, most of the probability mass is typically contained in a few peaks of the posterior distribution. Model selection means using only the most massive peaks and discarding the remaining models.

4 Ensemble learning

Ensemble learning, [1], is a recently introduced method for parametric approximation of the posterior distributions where Kullback-Leibler information, [2], [5], is used to measure the misfit between the actual posterior distribution and its approximation. Let us denote the observed variables by x and the unknown variables by y. The true posterior distribution $p(y|x)$ is approximated with the distribution $q(y|x)$ by minimising the Kullback-Leibler information.

$$\begin{aligned} D(q(y|x)||p(y|x)) &= \int q(y|x) \ln \frac{q(y|x)}{p(y|x)} dy \\ &= \int q(y|x) \ln \frac{p(x)q(y|x)}{p(x,y)} dy \\ &= \int q(y|x) \ln \frac{q(y|x)}{p(x,y)} dy + \ln p(x) \end{aligned} \tag{4}$$

The Kullback-Leibler information is greater than or equal to zero, with equality if and only if the two distributions, $p(y|x)$ and $q(y|x)$, are equivalent. Therefore the Kullback-Leibler information acts as a distance measure between the two distributions.

If we note that the term $p(x)$ is a constant over all the models, we can define a cost function $C_y(x)$ which we are required to minimise to obtain the optimum approximating distribution

$$C_y(x) = D(q(y|x)||p(y|x)) - \ln p(x) = \int q(y|x) \ln \frac{q(y|x)}{p(x,y)} dy \tag{5}$$

We shall adopt the notation that the subindex of C denotes the variables that are marginalised over in the cost function. In general, they are the unknown variables of the model. The notation also makes explicit that the cost function $C_y(x)$ gives an upper bound for $-\ln p(x)$. Here we use the same notation as with probability distributions, that is $C_y(x|z)$ means

$$\begin{aligned} C_y(x|z) &= D(q(y|x,z)||p(y|x,z)) - \ln p(x|z) \\ &= \int q(y|x,z) \ln \frac{q(y|x,z)}{p(x,y|z)} dy \end{aligned} \tag{6}$$

and thus yields the upper bound for $-\ln p(x|z)$.

Ensemble learning is practical if the terms $p(x,y)$ and $q(y|x)$ of the cost function $C_y(x)$ can be factorised into simple terms. If this is the case, the logarithms in the cost function split into sums of many simple terms. By virtue of the definition of the models, the

likelihood and priors are both likely to be products of simpler distributions, $p(x,y)$ typically factorises into simple terms. In order to simplify the approximating ensemble, q is also modelled as a product of simple terms.

The Kullback-Leibler information is a global measure, providing that the approximating distribution is a global distribution. Therefore the measure will be sensitive to probability mass in the true posterior distribution rather than the absolute value of the distribution itself.

Training the approximating ensemble can be done by assuming a fixed parametric form for the ensemble (for instance assuming a product of Gaussians). The parameters of the distributions can then be set to minimise the cost function.

An alternative method is to only assume a separable form for the approximating ensemble. The distributions themselves can then be found by performing a functional minimisation of the cost function with respect to each distribution in the ensemble. While this method must always give ensembles with equivalent or lower misfits than those obtained by a assuming a parametric form, the distributions that are obtained are not always tractable and so the fixed form method may be more useful.

4.1 Model selection in ensemble learning

Recall that the cost function $C_y(x|H)$ can be translated into lower bound for $p(x|H)$. Since $p(H|x) = p(x|H)p(H)/p(x)$, it is natural that $C_y(x|H)$ can be used for model selection also by equating

$$p(H|x) \approx \frac{e^{-C_y(x|H)}P(H)}{\sum_{H'} e^{-C_y(x|H')}P(H')} . \tag{7}$$

In fact, we can show that the above equation gives the best approximation for $p(H|x)$ in terms of $C_{y,H}(x)$, the Kullback-Leibler divergence between $q(y,H|x)$ and $p(y,H|x)$, which means that the model selection can be done using the same principle of approximating the posterior distribution as learning parameters.

Without losing any generality from $q(y,H|x)$, we can write

$$q(y,H|x) = Q(H|x)q(y|x,H) . \tag{8}$$

Now the cost function can be written as

$$\begin{aligned} C_{y,H}(x) &= \sum_H \int q(y,H|x) \ln \frac{q(y,H|x)}{p(x,y,H)} dy \\ &= \sum_H Q(H|x) \int q(y|x,H) \ln \frac{Q(H|x)q(y|x,H)}{P(H)p(x,y|H)} dy \\ &= \sum_H Q(H|x) \left[\ln \frac{Q(H|x)}{P(H)} + C_y(x|H) \right] . \end{aligned} \tag{9}$$

Minimising $C_{y,H}(x)$ with respect to $Q(H|x)$ under the constraint

$$\sum_H Q(H|x) = 1 \tag{10}$$

yields

$$Q(H|x) = \frac{e^{-C_y(x|H)}P(H)}{\sum_{H'} e^{-C_y(x|H')}P(H')} . \quad (11)$$

Substituting this into equation 9 yields the minimum value for $C_{y,H}(x)$ which is

$$C_{y,H}(x) = -\ln \sum_H e^{-C_y(x|H)}P(H) \quad (12)$$

If we wish to use only a part of different model structures H, we can try to find those H which would minimise $C_{y,H}(x)$. It is easy to see that this is accomplished by choosing the models corresponding to $C_y(x|H)$. A special case is to use only one H corresponding to the smallest $C_y(x|H)$.

4.2 Connection to coding

The cost function can be derived within MDL framework. The intuitive idea behind MDL is that in order to be able to encode data compactly one has to find a good model for it. Coding is not actually done; only the formula for the code length is needed. The optimal code length of x is $L(x) = -\log P(x)$ bits, and therefore coding has a close connection to the probabilistic framework. Some people prefer to think in terms of coding lengths, some in term of probabilities.

In coding, the sender and the receiver have to agree on the structure of the message in advance. This corresponds to having to state one's priors in the Bayesian framework.

A simple example: let x be the observation to be coded and y be a real valued parameter. We first encode y with accuracy dy. This part of the message takes $L(y) = -\log p(y)dy$ bits. Then we encode x with accuracy dx using the value y. This takes $L(x|y) = -\log p(x|y)dx$ bits. The accuracy of the encoding of the observations can be agreed on in advance, but there is a question of how to decide the accuracy dy. The second part of the code would be shortest if y would be exactly in the maximum likelihood solution. If y is encoded in too high accuracy, however, the first part of the code will be very long. If y is encoded in too low accuracy, the first part will be short but deviations from the optimal y will increase the second part of the code.

The bits-back argument, [1], overcomes the problem by using infinitesimally small dy but picking the y from a distribution $q(y)$ and encoding a secondary message in the choice of y. Since the distribution $q(y)$ is not needed for decoding x, both the sender and the receiver can run the same algorithm for determining $q(y)$ from x. After finding out $q(y)$, the receiver can decode the secondary message which can have $-\log q(y)dy$ bits. As these bits will be got back in the end, the expected message length is

$$\begin{aligned} L(x) &= E_q\{-\log p(x|y)dx - \log p(y)dy + \log q(y)dy\} \\ &= D(q(y)||p(y|x)) - \log p(x)dx = C_y(x) - \log dx . \quad (13) \end{aligned}$$

The amount of bits used for individual parameter θ can be measured by looking at $D(q(\theta)||p(\theta))$

It is interesting to notice that although the message has two parts, y and x, if $q(y)$ is chosen to be $p(y|x)$, the expected code length is equal to that of the optimal one-part message where the sender encodes x directly using the marginal probability $p(x) = \int p(x|y)p(y)dy$.

4.3 EM and MAP

Expectation maximisation (EM) algorithm can be seen as a special case of ensemble learning. The set-up in EM is the following: Suppose we have a probability model $p(x, y|\theta)$. We observe x but y remains hidden. We would like to estimate θ with maximum likelihood, i.e., maximise $p(x|\theta)$ w.r.t. θ, but suppose the structure of the model is such that integration over $p(x, y|\theta)$ is difficult, i.e., it is difficult to evaluate $p(x|\theta) = \int p(x, y|\theta)dy$.

What we do is take the cost function $C_y(x|\theta)$ and minimise it alternately with respect to θ and $q(y|x, \theta)$. The ordinary EM algorithm will result when $q(y|x, \theta)$ has a free form in which case $q(y|x, \theta)$ will be updated to be $p(y|x, \hat{\theta})$, where $\hat{\theta}$ is the current estimate of θ. The method is useful if integration over $\ln p(x, y|\theta)$ is easy, which is often the case. This interpretation of EM was given by [4].

EM algorithm can suffer from overfitting because only point estimates for the parameters θ are used. Even worse is to use maximum a posterior (MAP) estimator where one finds the θ and y which maximise $p(y, \theta|x)$. Unlike maximum likelihood estimation, MAP estimation is not invariant under reparametrisations of the model. This is because MAP estimation is sensitive to probability density which changes nonuniformly if the parameter space is changed nonlinearly.[1]

MAP estimation can be interpreted in ensemble learning framework as minimising $C_{y,\theta}(x)$ and using delta-distribution as $q(y, \theta|x)$. This makes the integral $\int q(y, \theta|x) \ln q(y, \theta|x)dyd\theta$ infinite. It can be neglected when estimating θ and y because it is constant with respect to $\hat{y}$ and $\hat{\theta}$, but the infinity of the cost function shows that delta distribution, i.e. a point estimator, is a bad approximation for a posterior density.

5 Construction of probabilistic models

In order to apply the Bayesian approach for modelling, the model needs to be given in probabilistic terms, which means stating the joint distribution of all the variables in the model. In principle, any joint distribution can be regarded as a model, but in practice, the joint distribution will have a simple form.

As an example, we shall see how a generative model turns into a probabilistic model. Suppose we have a model which tells how a sequence $\vec{y} = y(1), \ldots , y(t)$ transforms into sequence $\vec{x} = x(1), \ldots , x(t)$.

$$x(t) = f(y(t), \theta) + n(t) \quad (14)$$

This is called a generative model for $\vec{x}$ because it tells explicitly how the squence $\vec{x}$ is generated from the sequence $\vec{y}$ through a mapping f parametrised by θ. As it is usually unrealistic to assume that it would be possible to model all the things affecting $\vec{x}$ exactly, the models typically include a noise term $n(t)$.

If $y(t)$ and θ are given, then $x(t)$ has the same distribution as $n(t)$ except that it is offset by $f(y(t), \theta)$. This means that if $n(t)$ is Gaussian noise with variance σ^2, equation 14 translates into

$$x(t) \sim N(f(y(t)), \sigma^2) \quad (15)$$

[1]MAP estimation can be made invariant under reparametrisations by fixing the parametrisation. A natural way to do this is to use a parametrisation which makes the Fisher information matrix of the model constant. Then it is probable that the peaks in the posterior have approximately similar widths and are more or less symmetrical.

which is equivalent to

$$p(x(t)|y(t),\theta,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{[x(t)-f(y(t),\theta)]^2}{2\sigma^2}} . \tag{16}$$

The joint density of all the variables can then be written as

$$p(\vec{x},\vec{y},\theta,\sigma) = p(\vec{y},\theta,\sigma) \prod_t p(x(t)|y(t),\theta,\sigma) . \tag{17}$$

Usually also the probability $p(\vec{y},\theta,\sigma)$ is stated in a factorisable form making the full joint probability density $p(\vec{x},\vec{y},\theta,\sigma)$ a product of many simple terms.

In supervised learning, the sequence $\vec{y}$ is assumed to be fully known, also for any future data, which means that the full joint probability $p(\vec{x},\vec{y},\theta,\sigma)$ is not needed, only

$$p(\vec{x},\theta,\sigma|\vec{y}) = p(\vec{x}|\vec{y},\theta,\sigma)p(\theta,\sigma|\vec{y}) . \tag{18}$$

Typically $\vec{y}$ is assumed to be independent of θ and σ, i.e., $p(\vec{y},\theta,\sigma) = p(\vec{y})p(\theta,\sigma)$. This also means that $p(\theta,\sigma|\vec{y}) = p(\theta,\sigma)$ and thus only $p(\vec{x}|\vec{y},\theta,\sigma)$, given by the generative model equation 14, and the prior for the parameters $p(\theta,\sigma)$ is needed in supervised learning.

If the probability $p(\vec{y})$ is not modelled in supervised learning, it is impossible to treat missing elements of the sequence $\vec{y}$. If the probability $p(\vec{y})$ is modelled, however, there are no problems. The posterior density is computed for all unknown variables, including the missing elements of $\vec{y}$. In fact, unsupervised learning can be seen as a special case where the whole sequence $\vec{y}$ is unknown. In probabilistic framework, the treatment of any missing values is possible as long as the model defines the joint density of all the variables in the model. It is, for instance, easy to treat missing elements of sequence $\vec{x}$ or mix freely between supervised and unsupervised learning depending on how large part of the sequence $\vec{y}$ is known.

5.1 Priors and hyperpriors

In the above model, the parameters θ and σ need to be assigned a prior probability and in a more general situation typically also a prior probability for the model structure would also be needed.

In general, prior probabilities for variables should reflect the belief one has about the variables. As people may have difficulties in articulating these beliefs explicitly, some rules of thumb have been developed.

A lot of research has been conducted on uninformative priors. The term means that in some sense the prior gives as little information as possible about the value of the parameter, and as such is a good reference prior although with complex models it is usually impractical to compute the exact form of the uninformative prior.

Roughly speaking, uninformative prior can be defined by saying that in a parametrisation where moving a given distance in parameter space always corresponds the similar change in the probability distribution the model defines, the parameters are uniformly distributed.

A simple example is provided by a Gaussian distribution parametrised by mean μ and variance σ^2. Doubling the variance always results in a qualitatively similar change in the distribution. Similarly, taking a step of size σ in the mean always corresponds to a similar change in the distribution. Reparametrisation by $\mu' = \mu/\sigma$ and $v = \ln \sigma$ will

give a parametrisation where equal changes in parameters correspond to equal changes in distribution. A uniform prior on μ' and v would correspond to the prior $p(\mu,\sigma) \propto 1/\sigma^2$ in the original parameter space. If there is additional knowledge that μ sould be independent of σ, then μ and v give the needed parameters and the prior is $p(\mu) \propto 1$ and $p(\sigma) \propto 1/\sigma$.

None of the above uninformative priors can actually be used because they are improper, meaning that they are not normalisable. This can easily be seen by considering a uniform distribution between $\pm\infty$. These priors are nevertheless good references and hint useful parametrisations for models.

Often it is possible to utilise the fact that the model has a set of parameters which have a similar role. It is, for instance, reasonable to assume that all biases in an MLP network have a similar distribution. This knowledge can be utilised by modelling the distribution by a common parametrised distribution. Then the prior needs to be determined to these common parameters, called hyperparameters but as they control the distribution of a set of parameters, there should be less hyperparameters than parameters. The process can be iterated until the structural knowledge has been used. In the end there are usually only a few priors to determine and since there is typically a lot of data, these priors are usually not significant for the leaining process.

For some it may be helpful to think about the prior in terms of coding. By using the formula $L(x) = -\ln p(x)$, any probabilites can be translated into encoding. In coding terms, the prior means the aspects of the encoding which the sender and the receiver have agreed upon prior to the transmission of data.

6 Examples

6.1 Fixed form Q

Let us model a set of observations, $\vec{x} = x(1),\ldots,x(t)$, by a Gaussian distribution parametrised by mean m and log-std $v = \ln\sigma$. We shall approximate the posterior distribution by $q(m,v) = q(m)q(v)$, where both $q(m)$ and $q(v)$ are Gaussian. The parametrisation with log-std is chosen because the posterior of v is closer to Gaussian than the posterior of σ or σ^2 would be. (Notice that the parametrisation yielding close to Gaussian posterior distributions is connected to uninformative priors discussed in section 5.1.)

Let the priors for m and v be Gaussian with means μ_m and μ_v and variances σ_m^2 and σ_v^2, respectively. The joint density of the observations and the parameters m and v is

$$p(\vec{x},m,v) = \left[\prod_t p(x(t)|m,v)\right] p(m)p(v) . \tag{19}$$

As we can see, the posterior is a product of many simple terms.

Let us denote by $\bar{m}$ and $\tilde{m}$ the posterior mean and variance of m.

$$q(m;\bar{m},\tilde{m}) = \frac{1}{\sqrt{2\pi\tilde{m}}} e^{-\frac{(m-\bar{m})^2}{2\tilde{m}}} \tag{20}$$

The distribution $q(v;\bar{v},\tilde{v})$ is analogous.

The cost function is now

$$\begin{aligned} C_{m,v}(\vec{x}) &= \int q(m,v) \ln \frac{q(m,v)}{p(\vec{x},m,v)} dmdv = \\ & \int q(m,v) \ln q(m) dmdv + \int q(m,v) \ln q(v) dmdv + \\ & -\sum_{t=1}^{N} \int q(m,v) \ln p(x(t)|m,v) dmdv + \\ & -\int q(m,v) \ln p(m) dmdv - \int q(m,v) \ln p(v) dmdv \,. \end{aligned} \tag{21}$$

We see that the cost function has many terms, all of which are expectations over $q(m,v)$. Since the approximation $q(m,v)$ is assumed to be factorised into $q(m,v) = q(m)q(v)$, it is fairly easy to compute these expectations. For instance, integrating the term $\int q(m,v) \ln q(m) dmdv$ over v yields $\int q(m) \ln q(m) dm$, since $\ln q(m)$ does not depend on v and. Since $q(m)$ is assumed to be Gaussian with mean $\bar{m}$ and variance $\tilde{m}$, this integral is, in fact, minus the entropy of a Gaussian distribution and we have

$$\int q(m) \ln q(m) dm = -\frac{1}{2}(1 + \ln 2\pi\tilde{m}) \,. \tag{22}$$

A similar term, with $\tilde{m}$ replaced by $\tilde{v}$, comes from $\int q(m,v) \ln q(v) dmdv$.

The terms where expectation is taken over $-\ln p(m)$ and $-\ln p(v)$ are also simple since

$$-\ln p(m) = \frac{1}{2} \ln 2\pi\sigma_m^2 + \frac{(m-\mu_m)^2}{2\sigma_2} \,, \tag{23}$$

which means that we only need to be able to compute the expectation of $(m-\mu_m)^2$ over the Gaussian $q(m)$ having mean $\bar{m}$ and variance $\tilde{m}$. This yields

$$\begin{aligned} E\{(m-\mu_m)^2\} &= E\{m^2\} - 2E\{m\mu_m\} + E\{\mu_m^2\} = \\ & \bar{m}^2 + \tilde{m} - 2\bar{m}\mu_m + \mu_m^2 = (\bar{m}-\mu_m)^2 + \tilde{m} \end{aligned} \tag{24}$$

since the variance can be defined by $\tilde{m} = E\{m^2\} - E\{m\}^2 = E\{m^2\} - \bar{m}^2$ which shows that $E\{m^2\} = \bar{m}^2 + \tilde{m}$. Integrating the equation 23 and substituting equation 24 thus yields

$$-\int q(m) \ln p(m) dm = \frac{1}{2} \ln 2\pi\sigma_m^2 + \frac{(\bar{m}-\mu_m)^2 + \tilde{m}}{2\sigma_2} \,. \tag{25}$$

A similar term, with m replaced by v, will be obtained from $-\int q(v) \ln p(v) dv$.

The last terms are of the form $-\int q(m,v) \ln p(x(t)|m,v) dmdv$. Again we will find out that the factorisation $q(m,v) = q(m)q(v)$ simplifies the computation of these terms. Recall that $x(t)$ was assumed to be Gaussian with mean m and variance e^{2v}. The term over which the expectation is taken is thus

$$-\ln p(x(t)|m,v) = \frac{1}{2} \ln 2\pi + v + (x(t)-m)^2 e^{-2v} \,. \tag{26}$$

The expectation over the term $(x(t)-m)^2 e^{-2v}$ is easy to compute since m and v are assumed to be posteriorly independent. This means that it is possible to take the expectation separately from $(x(t)-m)^2$ and e^{-2v}. Using a similar derivation as in equation 24 yields $(x(t)-\bar{m})^2 + \tilde{m}$ for the first term. The expectation over e^{-2v} is also fairly easy to compute:

$$\begin{aligned} \int q(v) e^{-2v} dv &= \frac{1}{\sqrt{2\pi\tilde{v}}} \int e^{-\frac{(v-\bar{v})^2}{2\tilde{v}}} e^{-2v} dv = \frac{1}{\sqrt{2\pi\tilde{v}}} \int e^{-\frac{(v-\bar{v})^2 + 4v\tilde{v}}{2\tilde{v}}} dv = \\ & \frac{1}{\sqrt{2\pi\tilde{v}}} \int e^{-\frac{v^2 - 2v\bar{v} + \bar{v}^2 + 4v\tilde{v}}{2\tilde{v}}} dv = \\ & \frac{1}{\sqrt{2\pi\tilde{v}}} \int e^{-\frac{[v + (2\tilde{v} - \bar{v})]^2 + 4\bar{v}\tilde{v} - 4\tilde{v}^2}{2\tilde{v}}} = \\ & \frac{1}{\sqrt{2\pi\tilde{v}}} \int e^{-\frac{[v + (2\tilde{v} - \bar{v})]^2}{2\tilde{v}}} e^{2\tilde{v} - 2\bar{v}} dv = e^{2\tilde{v} - 2\bar{v}} \,. \end{aligned} \tag{27}$$

This shows that taking expectation over equation 26 yields a term

$$-\int q(m,v) \ln p(x(t)|m,v) dmdv = \frac{1}{2} \ln 2\pi + \bar{v} + [(x(t)-\bar{m})^2 + \tilde{m}] e^{2\tilde{v} - 2\bar{v}} \,. \tag{28}$$

Collecting together all the terms, we obtain the following cost function

$$\begin{aligned} C_{m,v}(\vec{x}; \bar{m}, \tilde{m}, \bar{v}, \tilde{v}) &= \sum_{t=1}^{N} \frac{1}{2} [(x(t)-\bar{m})^2 + \tilde{m}] e^{2\tilde{v} - 2\bar{v}} + N\bar{v} + \\ & \frac{(\bar{m}-\mu_m)^2 + \tilde{m}}{2\sigma_m^2} + \frac{(\bar{v}-\mu_v)^2 + \tilde{v}}{2\sigma_v^2} + \ln \sigma_m \sigma_v + \\ & \frac{N}{2} \ln 2\pi - \frac{1}{2} \ln \tilde{m}\tilde{v} - 1 \end{aligned} \tag{29}$$

Assuming σ_m^2 and σ_v^2 are very large, the minimum of the cost function can be solved by setting the gradient of the cost function C to zero. This yields the following:

$$\bar{m} = \frac{1}{N} \sum_t x(t) \tag{30}$$

$$\tilde{m} = \frac{\sum_t (x(t)-\bar{m})^2}{N(N-1)} \tag{31}$$

$$\bar{v} = \frac{1}{2N} + \frac{1}{2} \ln \frac{1}{N-1} \sum_t (x(t)-\bar{m})^2 \tag{32}$$

$$\tilde{v} = \frac{1}{2N} \tag{33}$$

In case σ_m^2 and σ_v^2 cannot be assumed very large, the equations for $\bar{v}$ and $\tilde{v}$ are not that simple, but the solution can still be obtained by solving the zero-point of the gradient.

Figure 3 shows a comparison of the true posterior distribution and the approximate posterior. The data set consisted of 100 points drawn from a model with $m = 1$ and $\sigma = 0.1$. The contours in both distributions are centred in the same region, a model that underestimates m.The contours for the two distributions are qualitatively similar although the true distribution is not symmetrical about the mean value of v.

6.2 Free Form Q

Instead of assuming the form for the approximate posterior distribution we could instead derive the optimal separable distribution (the functions that minimise the cost function subject to the constraint that they be normalised).

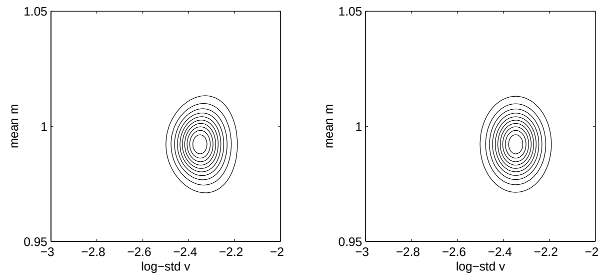

Figure 3: Comparison of the true and approximate posterior distributions for a test set containing 100 data points drawn from a model with $m = 1$ and $\sigma = 0.1$. The plot on the left shows the true posterior distribution over m and v. The plot on the right shows the approximate posterior distribution consisting of a diagonal Gaussian distribution.

Instead of learning the log-std $v = \ln \sigma$, we shall learn the inverse noise variance $\beta = \sigma^{-2}$. The prior on β is assumed to be a Gamma distribution of the form

$$p(\beta) = \frac{1}{\Gamma(c_\beta)} b_\beta^{c_\beta} \beta^{(c_\beta - 1)} \exp(-b_\beta \beta) \tag{34}$$

Setting $b_\beta = c_\beta = 10^{-3}$ leads to a broad prior in $\ln \beta$, this is equivalent to assuming that σ_v is large in the log-std parametrisation.

The cost function that must be minimised is now

$$\begin{aligned} C &= D(q(x, m, \beta) || P(x|m, \beta)) - \ln P(m, \beta) \\ &= \int q(m, \beta) \ln \frac{q(m, \beta)}{P(x, m, \beta)} dm d\beta \end{aligned} \tag{35}$$

If we assume a separable posterior, that is $q(m, \beta) = q(m)q(\beta)$ and substitute our priors into the cost function we obtain

$$\begin{aligned} C &= \int q(m)q(\beta) \ln \frac{q(m)q(\beta)}{\left[\prod_t P(x(t)|m, \beta)\right] P(m)P(\beta)} dm d\beta \\ &= \int q(m)q(\beta) \left[\ln q(m) + \ln q(\beta) - \ln P(m) - \ln P(\beta)\right. \\ &\quad \left. - \sum_t \left(\frac{1}{2} \ln \frac{\beta}{2\pi} - \frac{\beta (x(t) - m)^2}{2}\right)\right] dm d\beta \end{aligned} \tag{36}$$

Assuming we know $q(\beta)$ we can integrate over β in C (dropping any terms that are independent of m) to obtain

$$C = \int q(m) \left[\ln q(m) - \ln P(m) - \sum_t \left(-\frac{\bar{\beta} (x(t) - m)^2}{2}\right)\right] dm \tag{37}$$

where $\bar{\beta}$ is the average value of β under the distribution $q(\beta)$. We can optimise the cost function with respect to $q(m)$ by performing a functional derivative.

$$\begin{aligned} \frac{\partial C}{\partial q(m)} &= 1 + \ln q(m) - \ln P(m) - \sum_t \left(-\frac{\bar{\beta} (x(t) - m)^2}{2}\right) + \lambda_m \\ &= 0 \end{aligned} \tag{38}$$

where λ_m is a Lagrange multiplier introduced to ensure that $q(m)$ is normalised. Rearranging we see that

$$\begin{aligned} \ln q(m) &= -1 - \frac{1}{2} \ln 2\pi\sigma_m^2 - \frac{(m - mu_m)^2}{2\sigma_m^2} \\ &= +\sum_t \left(-\frac{\bar{\beta} (x(t) - m)^2}{2}\right) + \lambda_m \end{aligned} \tag{39}$$

and so the approximate posterior distribution is a Gaussian with variance $\tilde{m} = \left(\sigma_m^{-2} + T\bar{\beta}\right)^{-1}$ and mean $\bar{m} = \tilde{m} \left(\frac{\mu_m}{\sigma_m^2} + \bar{\beta} \sum_t x(t)\right)$.

We can obtain the optimum form for $q(\beta)$ by marginalising the cost function over m and dropping terms independent of β.

$$\begin{aligned} C &= \int q(\beta) \left[\ln q(\beta) - \ln P(\beta)\right. \\ &\quad \left. - \sum_t \left(\frac{1}{2} \ln \beta - \frac{\beta \left((x(t) - \bar{m})^2 + \tilde{m}\right)}{2}\right)\right] d\beta \end{aligned} \tag{40}$$

Again we can perform a functional derivative to obtain

$$\begin{aligned} \frac{\partial C}{\partial q(\beta)} &= 1 + \ln q(\beta) - \ln P(\beta) \\ &\quad - \sum_t \left(\frac{1}{2} \ln \beta - \frac{\beta \left((x(t) - \bar{m})^2 + \tilde{m}\right)}{2}\right) + \lambda_\beta \\ &= 0 \end{aligned} \tag{41}$$

and so

$$\ln q(\beta) = -1 + \sum_t \left(\frac{1}{2} \ln \beta - \frac{\beta \left((x(t) - \bar{m})^2 + \tilde{m}\right)}{2}\right) + \ln P(\beta) + \lambda_\beta \tag{42}$$

So the optimal posterior distribution is a Gamma distribution, with parameters $\hat{b}_\beta = b_\beta + \frac{\sum_t \left((x(t) - \bar{m})^2 + \tilde{m}\right)}{2}$ and $\hat{c}_\beta = c_\beta + \frac{T}{2}$. Therefore the expectation of β under the posterior distribution is $\bar{\beta} = \frac{\hat{c}_\beta}{\hat{b}_\beta}$.

The optimal distributions for m and β depend on each other ($q(m)$ is a function of $\bar{\beta}$ and $q(\beta)$ is a function of $\bar{m}$ and $\tilde{m}$) so the optimal solutions can be found by iteratively updating $q(m)$ and $q(\beta)$.

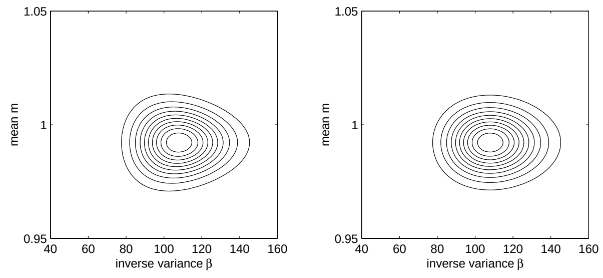

Figure 4: Comparison of the true and approximate posterior distributions for a test set containing 100 data points drawn from a model with $m = 1$ and $\sigma = 0.1$. The plot on the left shows the true posterior distribution over m and β. The plot on the right shows the approximate posterior distribution derived by obtaining the optimal free form separable distribution.

A general point is that the freeform optimisation of the cost function will typically lead to a set of iterative update equations where each distribution is updated on the basis of the other distributions in the approximation.

We can also see that if the parametrisation of the model is chosen appropriately the optimal separable model has a similar form to the prior model. If the prior distributions are Gaussians the posterior distributions are also Gaussians (likewise for Gamma distributions). If this is the case then we can say that we have chosen conjugate priors.

Figure 4 shows a comparison of the true posterior distribution and the approximate posterior. The data set is the same as for the fixed form example. The contours in both distributions are centred in the same region, a model that underestimates m. The contours for the two distributions are qualitatively similar, the approximate distribution also shows the assymmetric density.

7 Summary

In Ensemble Learning, the search for good models is sensitive to high probability mass and so the problems of over-fitting inherent to maximum likelihood and maximum posterior probability methods are removed.

The approximation of the posterior distribution assumes some degree of factorisation of the true distribution in order to make the approximation more tractable. Additionally the fixed form approximation also assumes some functional form of the factors.

It is often possible to affect the correctness of the approximation by the choice of parameterisation of the model. Also, the learning process tries to make the approximation more correct.

The free form approximation of the separable distribution will often result in Gaussian, Gamma, etc distributions if the parametrization of the model is chosen suitably. Therefore the optimisation process will suggest a parametrization for the problem.

References

[1] Geoffrey E. Hinton and Drew van Camp: 'Keeping neural networks simple by minimizing the description length of the weights'. In: *Proceedings of the COLT'93*, (Santa Cruz, California, 1993) pp. 5–13

[2] S. Kullback and R. A. Leibler: 'On information and sufficiency'. The Annals of Mathematical Statistics **22**, pp. 79–86 (1951)

[3] Radford M. Neal: 'Bayesian Learning for Neural Networks'. Lecture Notes in Statistics No. 118 (Springer-Verlag, 1996)

[4] Radford M. Neal and Geoffrey E. Hinton: 'A view of the EM algorithm that justifies incremental, sparse and other variants'. In: *Learning in Graphical Models.* ed. by Michael I. Jordan (1998)

[5] Claude E. Shannon: 'A mathematical theory of communication'. Bell System Technical Journal **27**, pp. 379–423 and 623–656 (1948)

[6] S. M. Stigler: 'Translation of Laplace's 1774 memoir on "Probability of causes"'. Statistical Science, **1(3)**, pp. 359–378 (1986)

BAYESIAN NONLINEAR INDEPENDENT COMPONENT ANALYSIS BY MULTI-LAYER PERCEPTRONS

Harri Lappalainen and Antti Honkela

In *Advances in Independent Component Analysis* (M. Girolami, ed.), pp. 93–121, Springer-Verlag, 2000.

Abstract

In this chapter, a nonlinear extension to independent component analysis is developed. The nonlinear mapping from source signals to observations is modelled by a multi-layer perceptron network and the distributions of source signals are modelled by mixture-of-Gaussians. The observations are assumed to be corrupted by Gaussian noise and therefore the method is more adequately described as nonlinear independent factor analysis. The nonlinear mapping, the source distributions and the noise level are estimated from the data. Bayesian approach to learning avoids problems with overlearning which would otherwise be severe in unsupervised learning with flexible nonlinear models.

1 Introduction

The linear principal and independent component analysis (PCA and ICA) model the data as having been generated by independent sources through a linear mapping. The difference between the two is that PCA restricts the distribution of the sources to be Gaussian, whereas ICA does not, in general, restrict the distribution of the sources.

In this chapter we introduce nonlinear counterparts of PCA and ICA where the generative mapping form sources to data is not restricted to be linear. The general form of the models discussed here is

$$\vec{x}(t) = f(\vec{s}(t)) + \vec{n}(t)\,. \tag{1}$$

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

Just as their linear counterparts, the nonlinear versions of PCA and ICA can be used for instance in dimension reduction and feature extraction. The difference between linear and nonlinear PCA is depicted in Fig. 1. In the linear PCA the data is described with a linear coordinate system whereas in the nonlinear PCA the coordinate system is nonlinear. The nonlinear PCA and ICA can be used for similar tasks as their linear counterparts, but they can be expected to capture the structure of the data better if the data points lie in a nonlinear manifold instead of a linear subspace.

Usually the linear PCA and ICA models do not have an explicit noise term $\vec{n}(t)$ and the model is thus simply

$$\vec{x}(t) = f(\vec{s}(t)) = \mathrm{A}\vec{s}(t) + \vec{b}\,. \tag{2}$$

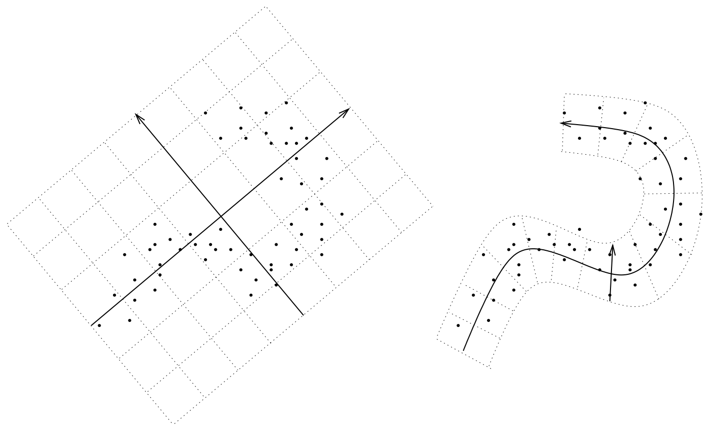

Figure 1: On the left hand side the data is described with a linear coordinate system. On the right hand side the coordinate system is nonlinear

The corresponding PCA and ICA models which include the noise term are often called factor analysis and independent factor analysis (FA and IFA) models. The nonlinear models discussed here can therefore also be called nonlinear factor analysis and nonlinear independent factor analysis models.

In this chapter, the distribution of sources is modelled with Gaussian density in PCA and mixture-of-Gaussians density in ICA. Given enough Gaussians in the mixture, any density can be modelled with arbitrary accuracy using the mixture-of-Gaussians density, which means that the source density model is universal. Likewise, the nonlinear mapping $f()$ is modelled by a multi-layer perceptron (MLP) network which can approximate any nonlinear mapping with arbitrary accuracy given enough hidden neurons.

The noise on each observation channel (component of data vectors) is assumed to be independent and Gaussian, but the variance of the noise on different channels is not assumed to be equal. The noise could be modelled with a more general distribution, but we shall restrict the discussion to the simple Gaussian case. After all, noise is supposed to be something uninteresting and unstructured. If the noise is not Gaussian or independent, it is a sign of interesting structure which should be modelled by the generative mapping from the sources.

2 Choosing Among Competing Explanations

Each model with particular values for sources, parameters and noise terms can be considered as an explanation for the observations. Even with linear PCA and ICA there are infinitely many possible models which explain the observations completely. With flexible nonlinear models like an MLP network, the number of possible explanations is — loosely speaking — even higher (although mathematically speaking, ∞^2 would still be 'only' ∞).

An example of competing explanations is given in Fig. 2. The data is sampled from

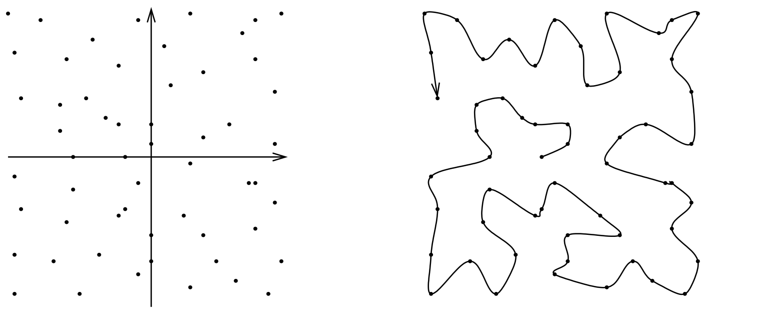

Figure 2: The data is generated by two independent evenly distributed sources as shown on the left. Given enough hidden neurons, an MLP network is able to model the data as having been generated by a single source through a very nonlinear mapping, depicted on the right

an even distribution inside a square. This is equivalent to saying that two independent sources, each evenly distributed, have generated the data as shown on the left hand side of the figure. If we only look at the probability of the data, the nonlinear mapping depicted on the right hand side of the figure is even better explanation as it gives very high probabilities to exactly those data points that actually occurred. However, it seems intuitively clear that the nonlinear model in Fig. 2 is much more complex than the available data would justify.

The exact Bayesian solution is that instead of choosing a single model, all models are used by weighting them according to their posterior probabilities. In other words, each model is taken into account in proportion with how probable they seem in light of the observations.

If we look at the predictions the above two models give about future data points, we notice that the more simple linear model with two sources predicts new points inside the square but the more complex nonlinear model with one source predicts new points only along the curved line. The prediction given by the more simple model is evidently closer to the prediction obtained by the exact Bayesian approach where the predictions of all models would be taken into account by weighting them according to the posterior probabilities of the models.

With complex nonlinear models like MLP networks, the exact Bayesian treatment is computationally intractable and we are going to resort to ensemble learning, which is discussed in Chap. 6[1]. In ensemble learning, a computationally tractable parametric approximation is fitted to the posterior probabilities of the models.

In Sect. 6.4.2[2] it is shown that ensemble learning can be interpreted as finding the most simple explanation for the observations[3]. This agrees with the intuition that in Fig. 2, the simple linear model is better than the complex nonlinear model.

[1]Refers to publication IV.
[2]Refers to section 4.2 of publication IV.
[3]The complexity of the explanation is defined as the number of bits it takes to encode the observation using the model. In this case one would measure the total code length of the sources $s(\vec{t})$, the parameters of the mapping and the noise $n(\vec{t})$.

The fact that we are interested in simple explanations also explains why nonlinear ICA is needed at all if we can use nonlinear PCA. The nonlinearity of the mapping allows the PCA model to represent any time-independent probability density of the observations as originating from independent sources with Gaussian distributions. It would therefore seem that the non-Gaussian source models used in the nonlinear ICA cannot further increase the representational power of the model. However, for many naturally occurring processes the representation with Gaussian sources requires more complex nonlinear mappings than the representation with mixtures-of-Gaussians. Therefore the nonlinear ICA will often find a better explanation for the observations than the nonlinear PCA.

Similar considerations also explain why to use the MLP network for modelling the nonlinearity. Experience has shown that with MLP networks it is easy to model fairly accurately many naturally occurring multidimensional processes. In many cases the MLP networks give a more simple parametrisation for the nonlinearity than, for example, Taylor or Fourier series expansions.

On the other hand, it is equally clear that the ordinary MLP networks with sigmoidal nonlinearities are not the best models for all kinds of data. With the ordinary MLP networks it is, for instance, difficult to model mappings which have products of the sources. The purpose of this chapter is not to give the ultimate model for any data but rather to give a good model for many data, from which one can start building more sophisticated models by incorporating domain-specific knowledge. Most notably, the source models described here do not assume time-dependencies, which are often significant.

3 Nonlinear Factor Analysis

This section introduces a nonlinear counterpart of principal component analysis. As explained in Sect. 1, the model includes a noise term and we shall therefore call it nonlinear factor analysis. Learning is based on Bayesian ensemble learning which is introduced in Chap. 6[4]. In order to keep the derivations simple, only Gaussian probability distributions are used which allows us to utilise many of the formulas derived in Sect. 6.6.1[5].

The posterior probability density of the unknown variables is approximated by a Gaussian distribution. As in Chap. 6[6], the variances of the Gaussian distributions of the model are parametrised by logarithm of standard deviation, log-std, because then the posterior distribution of these parameters will be closer to Gaussian which then agrees better with the assumption that the posterior is Gaussian.

3.1 Definition of the Model

The schematic structure of the mapping is shown in Fig. 3. The nonlinearity of each hidden neuron is the hyperbolic tangent, which is the same as the usual logistic sigmoid except for a scaling. The equation defining the mapping is

$$\vec{x}(t) = f(\vec{s}(t)) + \vec{n}(t) = \mathrm{B}\tanh(\mathrm{A}\vec{s}(t) + \vec{a}) + \vec{b} + \vec{n}(t)\,. \tag{3}$$

The matrices A and B are the weights of first and second layer and $\vec{a}$ and $\vec{b}$ are the corresponding biases.

[4]Refers to publication IV.
[5]Refers to section 6.1 of publication IV.
[6]Refers to publication IV.

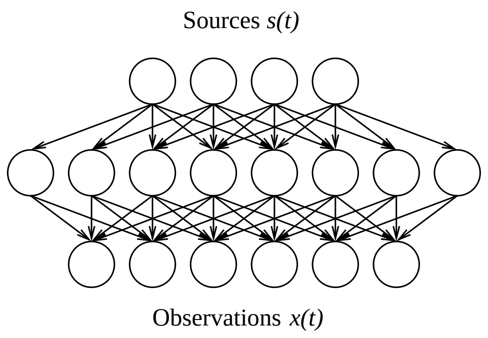

Figure 3: The mapping from sources to observations is modelled by the familiar MLP network. The sources are on the top layer and observations in the bottom layer. The middle layer consists of hidden neurons each of which computes a nonlinear function of the inputs

The noise is assumed to be independent and Gaussian and therefore the probability distribution of $x(t)$ is

$$\vec{x}(t) \sim N(f(\vec{s}(t)), e^{2\vec{v}_x}) \tag{4}$$

Each component of the vector $\vec{v}_x$ gives the log-std of the corresponding component of $\vec{x}(t)$.

The sources are assumed to have zero mean Gaussian distributions and again the variances are parametrised by log-std $\vec{v}_s$.

$$\vec{s}(t) \sim N(0, e^{2\vec{v}_s}) \tag{5}$$

Since the variance of the sources can vary, variance of the weights A on the first layer can be fixed to a constant, which we choose to be one, without loosing any generality from the model. This is not case for the second layer weights. Due to the nonlinearity, the variances of the outputs of the hidden neurons are bounded from above and therefore the variance of the second layer weights cannot be fixed. In order to enable the network to shut off extra hidden neurons, the weights leaving one hidden neuron share the same variance parameter[7].

$$\mathrm{B} \sim N(0, e^{2\vec{v}_B}) \tag{6}$$

The elements of the matrix B are assumed to have a zero mean Gaussian distribution with individual variances for each column and thus the dimension of the vector $\vec{v}_B$ is the number of hidden neurons. Both biases $\vec{a}$ and $\vec{b}$ have Gaussian distributions parametrised by mean and log-std.

[7]A hidden neuron will be shut off if all leaving weights are close to zero. Thinking in coding terms, it is easier for the network to encode this in one variance parameter than to encode it independently for all the weights.

The distributions are summarised in (7)–(12).

$$\begin{aligned}
\vec{x}(t) &\sim N(f(\vec{s}(t)), e^{2\vec{v}_x}) && (7)\\
\vec{s}(t) &\sim N(0, e^{2\vec{v}_s}) && (8)\\
\mathrm{A} &\sim N(0, 1) && (9)\\
\mathrm{B} &\sim N(0, e^{2\vec{v}_B}) && (10)\\
\vec{a} &\sim N(m_a, e^{2v_a}) && (11)\\
\vec{b} &\sim N(m_b, e^{2v_b}) && (12)
\end{aligned}$$

The distributions of each set of log-std parameters are modelled by Gaussian distributions whose parameters are usually called hyperparameters.

$$\begin{aligned}
\vec{v}_x &\sim N(m_{v_x}, e^{2v_{v_x}}) && (13)\\
\vec{v}_s &\sim N(m_{v_s}, e^{2v_{v_s}}) && (14)\\
\vec{v}_B &\sim N(m_{v_B}, e^{2v_{v_B}}) && (15)
\end{aligned}$$

The prior distributions of m_a, v_a, m_b, v_b and the six hyperparameters $m_{v_s}, \dots, v_{v_B}$ are assumed to be Gaussian with zero mean and standard deviation 100, i.e., the priors are assumed to be very flat.

3.2 Cost Function

In ensemble learning, the goal is to approximate the posterior pdf of all the unknown values in the model. Let us denote the observations by X. Everything else in the model is unknown, i.e., the sources, parameters and hyperparameters. Let us denote all these unknowns by the vector $\vec{\theta}$. The cost function measures the misfit between the actual posterior pdf $p(\vec{\theta}|\mathrm{X})$ and its approximation $q(\vec{\theta}|\mathrm{X})$.

The posterior is approximated as a product of independent Gaussian distributions

$$q(\vec{\theta}|\mathrm{X}) = \prod_i q(\theta_i|\mathrm{X})\,. \tag{16}$$

Each individual Gaussian $q(\theta_i|\mathrm{X})$ is parametrised by the posterior mean $\bar{\theta}_i$ and variance $\tilde{\theta}_i$ of the parameter.

The functional form of the cost function $C_{\vec{\theta}}(\mathrm{X}; \vec{\bar{\theta}}, \vec{\tilde{\theta}})$ is given in Chap. 6[8]. The cost function can be interpreted to measure the misfit between the actual posterior $p(\vec{\theta}|\mathrm{X})$ and its factorial approximation $q(\vec{\theta}|\mathrm{X})$. It can also be interpreted as measuring the number of bits it would take to encode X when approximating the posterior pdf of the unknown variables by $q(\vec{\theta}|\mathrm{X})$.

The cost function is minimised with respect to the posterior means $\bar{\theta}_i$ and variances $\tilde{\theta}_i$ of the unknown variables θ_i. The end result of the learning is therefore not just an estimate of the unknown variables, but a distribution over the variables.

The simple factorising form of the approximation $q(\vec{\theta}|\mathrm{X})$ makes the cost function computationally tractable. The cost function can be split into two terms, C_q and C_p, where the former is an expectation over $\ln q(\vec{\theta}|\mathrm{X})$ and the latter is an expectation over $-\ln p(\mathrm{X}, \vec{\theta})$.

[8]Refers to publication IV.

It turns out that the term C_q is not a function of the posterior means $\bar{\theta}_i$ of the parameters, only the posterior variances. It has a similar term for each unknown variable.

$$C_q(\mathrm{X}; \vec{\tilde{\theta}}) = \sum_i -\frac{1}{2} \ln 2\pi e \tilde{\theta}_i \tag{17}$$

Most of the terms of C_p are also trivial. The Gaussian densities in (8)–(15) yield terms of the form

$$-\ln p(\theta) = \frac{1}{2}(\theta - m_\theta)^2 e^{-2v_\theta} + v_\theta + \frac{1}{2} \ln 2\pi \tag{18}$$

Since θ, m_θ and v_θ are independent in q, the expectation over q yields

$$\frac{1}{2}[(\bar{\theta} - \bar{m}_\theta)^2 + \tilde{\theta} + \tilde{m}_\theta] e^{2\tilde{v}_\theta - 2\bar{v}_\theta} + \bar{v}_\theta + \frac{1}{2} \ln 2\pi\,. \tag{19}$$

Only the term originating from (7) needs some elaboration. Equation (7) yields

$$-\ln p(x) = \frac{1}{2}(x - f)^2 e^{-2v_x} + v_x + \frac{1}{2} \ln 2\pi \tag{20}$$

and the expectation over q is

$$\frac{1}{2}[(x - \bar{f})^2 + \tilde{f}] e^{2\tilde{v}_x - 2\bar{v}_x} + \bar{v}_x + \frac{1}{2} \ln 2\pi \tag{21}$$

The rest of this section is dedicated on evaluating the posterior mean $\bar{f}$ and variance $\tilde{f}$ of the function f. We shall begin from the sources and weights and show how the posterior mean and variance can be propagated through the network yielding the needed posterior mean and variance of the function f at the output. The effect of nonlinearities g of the hidden neurons are approximated by first and second order Taylor's series expansions around the posterior mean. Apart from that, the computation is analytical.

The function f consists of two multiplications with matrices and a nonlinearity in between. The posterior mean and variance for a product $u = yz$ are

$$\bar{u} = E\{u\} = E\{yz\} = E\{y\}E\{z\} = \bar{y}\bar{z} \tag{22}$$

and

$$\tilde{u} = E\{u^2\} - \bar{u}^2 = E\{y^2 z^2\} - (\bar{y}\bar{z})^2 = \\ E\{y^2\}E\{z^2\} - \bar{y}^2\bar{z}^2 = (\bar{y}^2 + \tilde{y})(\bar{z}^2 + \tilde{z}) - \bar{y}^2\bar{z}^2\,, \tag{23}$$

given that y and z are posteriorly independent. According to the assumption of the factorising form of $q(\vec{\theta}|\mathrm{X})$, the sources and the weights are independent and we can use the above formulas. The inputs going to hidden neurons consist of sums of products of weights and sources, each posteriorly independent, and it is therefore easy compute the posterior mean and variance of the inputs going to the hidden neurons; both the means and variances of a sum of independent variables add up.

Let us now pick one hidden neuron having nonlinearity g and input ξ, i.e., the hidden neuron is computing $g(\xi)$. At this point we are not assuming any particular form of g although we are going to use $g(\xi) = \tanh \xi$ in all the experiments; the following derivation is general and can be applied to any sufficiently smooth function g.

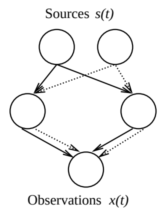

Figure 4: The converging paths from two sources are shown. Both input neurons affect the output neuron through two paths going through the hidden neurons. This means that the posterior variances of the two hidden neurons are neither completely correlated nor uncorrelated and it is impossible to compute the posterior variance of the output neuron without keeping the two paths separate. Effectively this means computing the Jacobian matrix of the output with respect to the inputs

In order to be able to compute the posterior mean and variance of the function g, we are going to apply the Taylor's series expansion around the posterior mean $\bar{\xi}$ of the input. We choose the second order expansion when computing the mean and the first order expansion when computing the variance. The choice is purely practical; higher order expansions could be used as well but these are the ones that can be computed from the posterior mean and variance of the inputs alone.

$$\bar{g}(\xi) \approx g(\bar{\xi}) + \frac{1}{2} g''(\bar{\xi})\tilde{\xi} \tag{24}$$

$$\tilde{g}(\xi) \approx [g'(\bar{\xi})]^2 \tilde{\xi} \tag{25}$$

After having evaluated the outputs of the nonlinear hidden neurons, it would seem that most of the work has already been done. After all, it was already shown how to compute the posterior mean and variance of a weighted sum and the outputs of the network will be weighted sums of the outputs of the hidden neurons. Unfortunately, this time the terms in the sum are no longer independent. The sources are posteriorly independent by virtue of the approximation $q(\vec{\theta}|\mathrm{X})$, but the values of the hidden neurons are posteriorly dependent which enforces us to use a more complicated scheme for computing the posterior variances of these weighted sums. The posterior means will be as simple as before, though.

The reason for the outputs of the hidden neurons to be posteriorly dependent is that the value of one source can potentially affect all the outputs. This is illustrated in Fig. 4. Each source affects the output of the whole network through several paths and in order to be able to determine the variance of the outputs, the paths originating from different sources need to be kept separate. This is done by keeping track of the partial derivatives $\frac{\partial g(\xi)}{\partial s_i}$. Equation (26) shows how the total posterior variance of the output $g(\xi)$ of one of the hidden neurons can be split into terms originating from each source plus a term $\tilde{g}^*(\xi)$ which contains the variance originating from the weights and biases, i.e., those variables which affect any one output through only a single path.

$$\tilde{g}(\xi) = \tilde{g}^*(\xi) + \sum_i \tilde{s}_i \left[\frac{\partial g(\xi)}{\partial s_i}\right]^2 \tag{26}$$

When the outputs are multiplied by weights, it is possible to keep track of how this affects the posterior mean, the derivatives w.r.t. the sources and the variance originating from other variables than the sources, i.e., from weights and biases. The total variance of the output of the network is then obtained by

$$\tilde{f} = \tilde{f}^* + \sum_i \tilde{s}_i \left[\frac{\partial f}{\partial s_i}\right]^2 , \tag{27}$$

where f denotes the components of the output and we have computed the posterior variance of the outputs of the network which is needed in (21). To recapitulate what is done, the contributions of different sources to the variances of the outputs of the network are monitored by computing the Jacobian matrix of the outputs w.r.t. the sources and keeping this part separate from the variance originating from other variables.

The only approximations done in the computation are the ones approximating the effect of nonlinearity. If the hidden neurons were linear, the computation would be exact. The nonlinearity of the hidden neurons is delt with by linearising around the posterior mean of the inputs of the hidden neurons. The smaller the variances the more accurate this approximation is. With increasing nonlinearity and variance of the inputs, the approximation gets worse.

Compared to ordinary forward phase of an MLP network, the computational complexity is greater by about a factor of $5N$, where N is the number of sources. The factor five is due to propagating distributions instead of plain values. The need to keep the paths originating from different sources separate explains the factor N. Fortunately, much of the extra computation can be made into good use later on when adapting the distributions of variables.

3.3 Update Rules

Any standard optimisation algorithm could be used for minimising the cost function $C(\mathrm{X}; \vec{\bar{\theta}}, \vec{\tilde{\theta}})$ with respect to the posterior means $\vec{\bar{\theta}}$ and variances $\vec{\tilde{\theta}}$ of the unknown variables. As usual, however, it makes sense utilising the particular structure of the function to be minimised.

Those parameters which are means or log-std of Gaussian distributions, e.g., m_b, m_{v_B}, v_a and v_{v_x}, can be solved in the same way as the parameters of Gaussian distribution where solved in Sect. 6.1. Since the parameters have Gaussian priors, the equations do not have analytical solutions, but Newton-iteration can be used. For each Gaussian, the posterior mean and variance of the parameter governing the mean is solved first by assuming all other variables constant and then the same thing is done for the log-std parameter, again assuming all other variables constant.

Since the mean and variance of the output of the network and thus also the cost function was computed layer by layer, it is possible to use the ordinary back-propagation algorithm to evaluate the partial derivatives of the part C_p of the cost function w.r.t. the posterior means and variances of the sources, weights and biases. Assuming the derivatives computed, let us first take a look at the posterior variances $\tilde{\theta}$.

The effect of the posterior variances $\tilde{\theta}$ of sources, weights and biases on the part C_p of the cost function is mostly due to the effect on $\tilde{f}$ which is usually very close to linear (this was also the approximation made in the evaluation of the cost function). The terms $\tilde{f}$ have a linear effect on the cost function, as is seen in (21), which means that the over all

effect of the terms $\tilde{\theta}$ on C_p is close to linear. The partial derivative of C_p with respect to $\tilde{\theta}$ is therefore roughly constant and it is reasonable to use the following fixed point equation to update the variances:

$$0 = \frac{\partial C}{\partial \tilde{\theta}} = \frac{\partial C_p}{\partial \tilde{\theta}} + \frac{\partial C_q}{\partial \tilde{\theta}} = \frac{\partial C_p}{\partial \tilde{\theta}} - \frac{1}{2\tilde{\theta}} \Rightarrow \tilde{\theta} = \frac{1}{2\frac{\partial C_p}{\partial \tilde{\theta}}} . \tag{28}$$

The remaining parameters to be updated are the posterior means $\bar{\theta}$ of the sources, weights and biases. For those parameters it is possible to use Newton iteration since the corresponding posterior variances $\tilde{\theta}$ actually contain the information about the second order derivatives of the cost function C w.r.t. $\bar{\theta}$. It holds

$$\tilde{\theta} \approx \frac{1}{\frac{\partial^2 C}{\partial \bar{\theta}^2}} \tag{29}$$

and thus the step in Newton iteration can be approximated

$$\bar{\theta} \leftarrow \bar{\theta} - \frac{\frac{\partial C_p}{\partial \bar{\theta}}}{\frac{\partial^2 C}{\partial \bar{\theta}^2}} \approx \bar{\theta} - \frac{\partial C_p}{\partial \bar{\theta}} \tilde{\theta} . \tag{30}$$

Equation (29) would be exact if the posterior pdf $p(\vec{\theta}|\mathrm{X})$ were exactly Gaussian. This would be true if the mapping f were linear. The approximation in (29) is therefore good as long as the function f is roughly linear around the current estimate of $\vec{\bar{\theta}}$.

3.3.1 Avoiding Problems Originating from Approximation of the Nonlinearity of the Hidden Neurons

The approximations in (24) and (25) can give rise to problems with ill defined posterior variances of sources or first layer weights A or biases $\vec{a}$. This is because the approximations take into account only local behaviour of the nonlinearities g of the hidden neurons. With MLP networks the posterior is typically multimodal and therefore, in a valley between two maxima, it is possible that the second order derivative of the logarithm of the posterior w.r.t. a parameter θ is positive. This means that the derivative of the C_p part of the cost function with respect to the posterior variance $\tilde{\theta}$ of that parameter is negative, leading to a negative estimate of variance in (28).

It is easy to see that the problem is due to the local estimate of g since the logarithm of the posterior eventually has to go to negative infinity. The derivative of the C_p term w.r.t. the posterior variance $\tilde{\theta}$ will thus be positive for large $\tilde{\theta}$, but the local estimate of g fails to account for this.

In order to discourage the network from adapting itself to areas of parameter space where the problems might occur and to deal with the problem if it nevertheless occurred, the terms in (24) which give rise to negative derivative of C_p with respect to $\tilde{\theta}$ will be neglected in the computation of the gradients. As this can only make the estimate of $\tilde{\theta}$ in (28) smaller, this leads, in general, to increasing the accuracy of the approximations in (24) and (25).

3.3.2 Stabilising the Fixed-Point Update Rules

The adaptations rules in (28) and (30) assume other parameters to be constant. The weights, sources and biases are updated all at once, however, because it would not be computationally efficient to update only one at a time. The assumption of independence is not necessarily valid, particularly for the posterior means of the variables, which may give rise to instabilities. Several variables can have a similar effect on outputs and when they are all updated to the values which would be optimal given that the others stay constant, the combined effect is too large.

This type of instability can be detected by monitoring the directions of updates of individual parameters. When the problem of correlated effects occurs, consecutive updated values start oscillating. A standard way to dampen these oscillations in fixed point algorithms is to introduce a learning parameter α for each parameter and update it according to the following rule:

$$\alpha \leftarrow \begin{cases} 0.8\alpha & \text{if sign of change was different} \\ \min(1, 1.05\alpha) & \text{if sign of change was same} \end{cases} \tag{31}$$

This gives the following fixed point update rules for the posterior means and variances of the sources, weights and the biases:

$$\bar{\theta} \quad \leftarrow \quad \bar{\theta} - \alpha_{\bar{\theta}} \frac{\partial C_p}{\partial \bar{\theta}} \tilde{\theta} \tag{32}$$

$$\tilde{\theta} \quad \leftarrow \quad \frac{1}{\left[2 \frac{\partial C_p}{\partial \tilde{\theta}}\right]^{\alpha_{\tilde{\theta}}}} \tilde{\theta}^{1-\alpha_{\tilde{\theta}}} \tag{33}$$

The reason why a weighted geometric rather than arithmetic mean is applied to the posterior variances is that variance is a scale parameter. The relative effect of adding a constant to the variance varies with the magnitude of the variance whereas the relative effect of multiplying by a constant is invariant.

3.3.3 Using Additional Information for Updating Sources

With sources, it is possible to measure and compensate some of the correlated effects in the updates. Recall that the Jacobian matrix of the output $\vec{f}$ of the network w.r.t. the sources was computed when taking into account the effects of multiple paths of propagating the values of sources. This will be used to compensate the assumption of independent updates, in addition to the learning rates α.

Suppose we have two sources whose effect on outputs are positively correlated. Assuming the effects independent means that the step will be too large and the actual step size should be less than what the Newton iteration suggests. This can be detected from computing the change resulting in the outputs and projecting it back for each source independently to see how much each source alone should change to produce the same change in the outputs. The difference between the change of one source in the update and change resulting from all the updates can then be used to adjust the step sizes in the update.

Two examples of correction are depicted in Fig. 5. The left hand side graph shows a case where the effects of sources on the outputs are positively correlated and the right hand side graph has negatively correlated effects. Current output of the network is in the origin O and the minimum of the cost function is in point A. Black arrows show where the output would move if the sources were minimised independently. The combined updates would then take the output to point B.

As the effects of sources on $\vec{x}$ are correlated, point B, the resulting overall change in $\vec{x}$, differs from point A. Projecting the point B back to the sources, comparison between

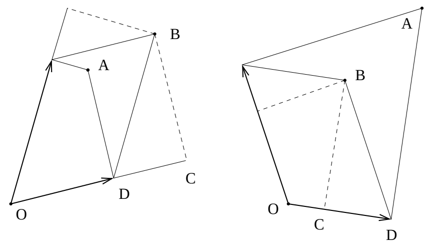

Figure 5: Illustration of the correction of error resulting from assuming independent updates of the sources. The figures show the effect two sources have on the outputs. On the left hand side the effects of sources on $\vec{x}$ are positively correlated and consequently the step sizes are overestimated. On the right hand side the effects are negatively correlated and the step sizes are underestimated

the resulting step size C and the desired step size D can be used for adjusting the step size. The new step size on the source would be D/C times the original. With positively correlated effects the adjusting factor D/C is less then one, but with negatively correlated sources it is greater than one. For the sake of stability, the corrected step is restricted to be at most twice the original.

4 Nonlinear Independent Factor Analysis

The nonlinear factor analysis model introduced in the previous section has Gaussian distributions for the sources. In this section we are going to show how that model can easily be extended to have mixture-of-Gaussians models for sources. In doing so we are largely following the method introduced in [1] for Bayesian linear independent factor analysis. The resulting model is a nonlinear counterpart of ICA or, more accurately, a nonlinear counterpart of independent factor analysis because the model includes finite noise. The difference between the models is similar to that between linear PCA and ICA because the first layer weight matrix A in the network has the same indeterminacies in nonlinear PCA as in linear PCA. The indeterminacy is discussed in the introductory chapter[9].

According to the model for the distribution of the sources, there are several Gaussian distributions and at each time instant, the source originates from one of them. Let us denote the index of the Gaussian from which the source $s_i(t)$ originates by $M_i(t)$. The model for the distribution for the ith source at time t is

$$p(s_i(t)|\theta) = \sum_{M_i(t)} P(M_i(t)|\theta) p(s_i(t)|\theta, M_i(t)) \,, \tag{34}$$

where $p(s_i(t)|\theta, M_i(t) = j)$ is a time-independent Gaussian distribution with its own mean m_{ij} and log-std v_{ij}. The probabilities $P(M_i(t)|\theta)$ of different Gaussians are modelled with

[9]Refers to the introductory chapter of *Advances in Independent Component Analysis.*

time-independent soft-max distributions.

$$P(M_i(t) = j|\theta) = \frac{e^{c_{ij}}}{\sum_{j'} e^{c_{ij'}}} \tag{35}$$

Each combination of different Gaussians producing the sources can be considered a different model. The number of these models is enormous, of course, but their posterior distribution can still be approximated by a similar factorial approximation which is used for other variables.

$$Q(\mathrm{M}|\mathrm{X}) = \prod_{M_i(t)} Q(M_i(t)|\mathrm{X}) \tag{36}$$

Without losing any further generality, we can now write

$$q(s_i(t), M_i(t)|\theta) = Q(M_i(t)|\theta)q(s_i(t)|\theta, M_i(t)), \tag{37}$$

which yields

$$q(s_i(t)|\theta) = \sum_j q(M_i(t) = j|\theta)Q(s_i(t)|\theta, M_i(t) = j). \tag{38}$$

This means that the approximating ensemble for the sources has a form similar to the prior, i.e., an independent mixture of Gaussians, although the posterior mixture is different at different times.

Due to the assumption of factorial posterior distribution of the models, the cost function can be computed as easily as before. Let us denote $Q(M_i(t) = j|\theta) = \dot{s}_{ij}(t)$ and the posterior mean and variance of $q(s_i(t)|\theta, M_i(t) = j)$ by $\bar{s}_{ij}(t)$ and $\tilde{s}_{ij}(t)$. It easy to see that the posterior mean and variance of $s_i(t)$ are

$$\bar{s}_i(t) = \sum_j \dot{s}_{ij}(t)\bar{s}_{ij}(t) \tag{39}$$

$$\tilde{s}_i(t) = \sum_j \dot{s}_{ij}(t)[\tilde{s}_{ij}(t) + (\bar{s}_{ij}(t) - \bar{s}_i(t))^2]. \tag{40}$$

After having computed the posterior mean $\bar{s}_i$ and variance $\tilde{s}_i$ of the sources, the computation of the C_p part of the cost function proceeds as with nonlinear factor analysis in the previous section. The C_q part yields terms of the form

$$\begin{aligned} & q(s_i(t)|\mathrm{X}) \ln q(s_i(t)|\mathrm{X}) = \\ & \quad \sum_j \dot{s}_{ij}(t)q(s_i(t)|M_i(t) = j, \mathrm{X}) \ln \sum_j \dot{s}_{ij}(t)q(s_i(t)|M_i(t), \mathrm{X}) = \\ & \quad \sum_j \dot{s}_{ij}(t)q(s_i(t)|M_i(t) = j, \mathrm{X}) \ln \dot{s}_{ij}(t)q(s_i(t)|M_i(t), \mathrm{X}) = \\ & \quad \sum_j \dot{s}_{ij}(t)[\ln \dot{s}_{ij}(t) + q(s_i(t)|M_i(t) = j, \mathrm{X}) \ln q(s_i(t)|M_i(t) = j, \mathrm{X})] \end{aligned} \tag{41}$$

and we have thus reduced the problem to a previously solved one. The terms $q(s_i(t)|M_i(t), \mathrm{X}) \ln q(s_i(t)|M_i(t), \mathrm{X})$ are the same as for the nonlinear factor analysis and

otherwise the equation has the same form as in model selection in Chap. 6[10]. This is not surprising since the terms $Q(M_i(t)|\mathrm{X})$ are the probabilities of different models and we are, in effect, therefore doing factorial model selection.

Most update rules are the same as for nonlinear factor analysis. Equations (39) and (40) bring the terms $\dot{s}_{ij}(t)$ for updating the means m_{ij} and log-std parameters v_{ij} of the sources. It turns out that they both will be weighted with $\dot{s}_{ij}$, i.e., the observation is used for adapting the parameters in proportion to the posterior probability of that observation originating from that particular Gaussian distribution.

5 Experiments

5.1 Learning Scheme

The learning scheme for all the experiments was the same. First, linear PCA is used to find sensible initial values for the posterior means of the sources. The method was chosen because it has given good results in initialising the model vectors of a self-organising map (SOM). The posterior variances of the sources are initialised to small values. Good initial values are important for the method since the network can effectively prune away unused parts as will be seen in the experiments later on. Initially the weights of the network have random values and the network has quite bad representation for the data. If the sources were adapted from random values also, the network would consider many of the sources useless for the representation and prune them away. This would lead to a local minimum from which the network would not recover.

Therefore the sources are fixed at the values given by linear PCA for the first 50 iterations through the whole data. This is long enough for the network to find a meaningful mapping from sources to the observations, thereby justifying using the sources for the representation. For the same reason, the parameters controlling the distributions of the sources, weights, noise and the hyperparameters are not adapted during the first 100 iterations. They are adapted only after the network has found sensible values for the variables whose distributions these parameters control.

In all simulations, the total number of iterations is 7500, where one iteration means going through all the observations. For nonlinear independent factor analysis simulations, a nonlinear subspace is estimated with 2000 iterations by the nonlinear factor analysis after which the sources are rotated with a linear ICA algorithm. In these experiments, FastICA was used [4]. The rotation of the sources is compensated by an inverse rotation to the first layer weight matrix A. The nonlinear independent factor analysis algorithm is then applied for the remaining 5500 iterations.

5.2 Helix

Let us first take a look at a toy problem which shows that it is possible to find a nonlinear subspace and model it with an MLP network in an unsupervised manner. A set of 1000 data points, shown on the left plot of Fig. 6, were generated from a normally distributed source s into a helical subspace. The z-axis had a linear correspondence to the source and the x- and y-axes were sine and cosine: $x = \sin(\pi s)$, $y = \cos(\pi s)$ and $z = s$. Gaussian noise with standard deviation 0.05 was added to all three data components.

[10]Refers to publication IV.

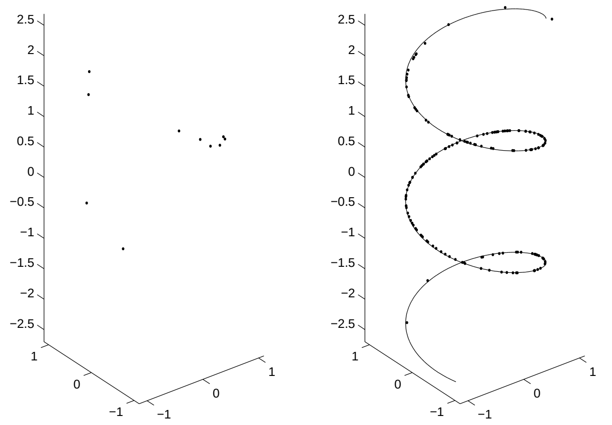

Figure 6: The plot on the left shows the data points and the plot on the right shows the reconstructions made by the network together with the underlying helical subspace. The MLP network has clearly been able to find the underlying one-dimensional nonlinear subspace where the data points lie

One-dimensional nonlinear subspaces were estimated with the nonlinear independent factor analysis algorithm. Several different numbers of hidden neurons and initialisations of the MLP networks were tested and the network which minimised the cost function was chosen. The best network had 16 hidden neurons. The original noisy data and the means of the outputs of the best MLP network are shown in Fig. 6. It is evident that the network was able to learn the correct subspace. Only the tails of the helix are somewhat distorted. The network estimated the standard deviations of the noise on different data components to be 0.052, 0.055 and 0.050. This is in close correspondence with the actual noise level of 0.05.

This problem is not enough to demonstrate the advantages of the method since it does not prove that the method is able to deal with high dimensional latent spaces. This problem was chosen simply because it is easy to visualise.

5.3 Nonlinear Artificial Data

5.3.1 Gaussian Sources

The following experiments with nonlinear factor analysis algorithm demonstrate the ability of the network to prune away unused parts. The data was generated from five normally distributed sources through a nonlinear mapping. The mapping was generated by a randomly initialised MLP network having 20 hidden neurons and ten output neurons. Gaussian noise with standard deviation of 0.1 was added to the data. The nonlinearity for the hidden neurons was chosen to be the inverse hyperbolic sine, which means that the nonlinear

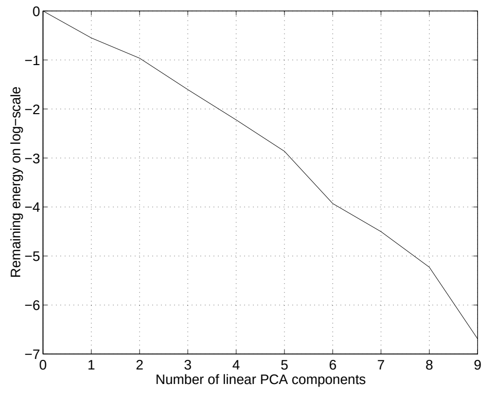

Figure 7: The graph shows the remaining energy in the data as a function of the number of extracted linear PCA components. The total energy is normalised to unity (zero on logarithmic scale). The data has been generated from five Gaussian sources but as the mapping is nonlinear, the linear PCA cannot be used for finding the original subspace

factor analysis algorithm using MLP network with tanh-nonlinearities cannot use exactly the same weights.

Figure 7 shows how much of the energy remains in the data when a number of linear PCA components are extracted. This measure is often used to deduce the linear dimension of the data. As the figure shows, there is no obvious turn in the curve and it would be impossible to deduce the nonlinear dimension.

With the nonlinear factor analysis by MLP networks, not only the number of the sources but also the number of hidden neurons in the MLP network needs to be estimated. With the Bayesian approach this is not a problem, as is shown in Figs. 8 and 9. The cost function exhibits a broad minimum as a function of hidden neurons and a saturating minimum as a function of sources. The reason why the cost function saturates as a function of sources is that the network is able to effectively prune away unused sources. In the case of ten sources, for instance, the network actually uses only five of them.

The pressure to prune away hidden neurons is not as big which can be seen in Fig. 10. A reliable sign of pruning is the amount of bits which the network uses for describing a variable. Recall that it was shown in Sect. 6.4.2 that the cost function can be interpreted as the description length of the data. The description length can also be computed for each variable separately and this is shown in Fig. 10. The MLP network had seven input neurons, i.e., seven sources, and 100 hidden neurons. The upper left plot shown clearly that the network effectively uses only five of the sources and very few bits are used to describe the other two sources. This is evident also from the first layer weight matrix A on the upper right plot, which shows the average description length of the weights leaving each input neuron.

The lower plot of Fig. 10 also shows the average description length of the weight matrix A, but now the average is taken row-wise and thus tells how many bits are used for

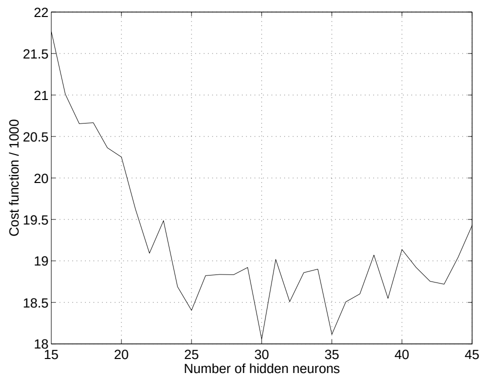

Figure 8: The value of the cost function is shown as a function of the number of hidden neurons in the MLP network modelling the nonlinear mapping from five sources to the observations. Ten different initialisations were tested to find the minimum value for each number of hidden neurons. The cost function exhibits a broad and somewhat noisy minimum. The smallest value for the cost function was obtained using 30 hidden neurons

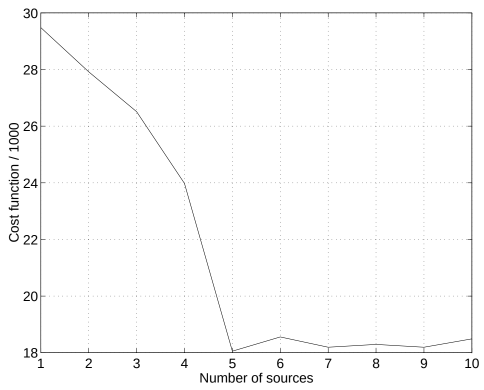

Figure 9: The value of the cost function is shown as a function of the number of sources. The MLP network had 30 hidden neurons. Ten different initialisations were tested to find the minimum value for each number of sources. The cost function saturates after five sources and the deviations are due to different random initialisation of the network

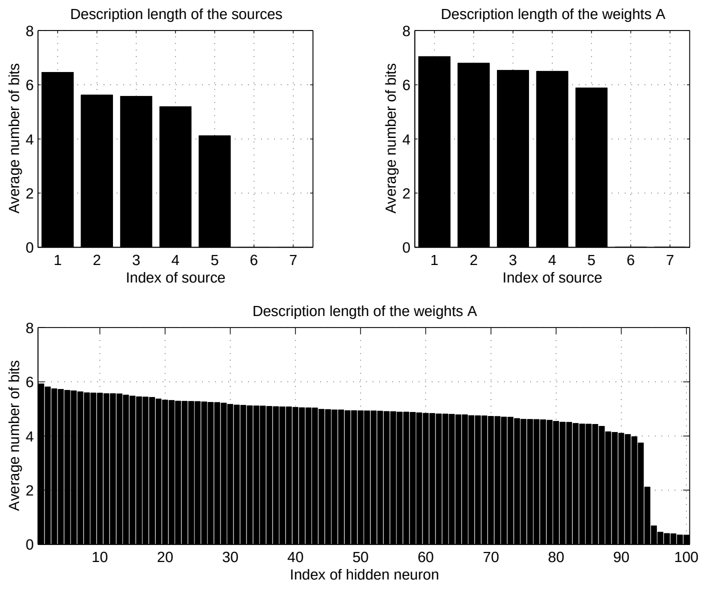

Figure 10: The network is able to prune away unused parts. This can be monitored by measuring the description length of different variables. The sources and hidden neurons are sorted by decreasing description length

describing the weights arriving to each hidden neuron. It appears that about six or seven hidden neurons have been pruned away, but the pruning is not as complete as in the case of sources. This is because for each source the network has to represent 1000 values, one for each observation vector, but for each hidden neuron the network only needs to represent five plus twenty (the effective number of inputs and outputs) values and there is thus much less pressure to prune away a hidden neuron.

5.3.2 Non-Gaussian Sources

The following experiments demonstrate the ability of the nonlinear independent factor analysis algorithm to find the underlying latent variables which have generated the observations.

In these experiments, a similar scheme was used to generate data as with the Gaussian sources before. Now a total of eight sources was used with four sub-Gaussian and four super-Gaussian sources. The generating MLP network was also larger, having 30 hidden neurons and 20 output neurons. The super-Gaussian sources were generated by taking a hyperbolic sine, $\sinh x$, from a normally distributed random variable and then normalising the variance to unity. In generating sub-Gaussian sources, inverse hyperbolic sine, $\sinh^{-1} x$, was used instead of $\sinh x$.

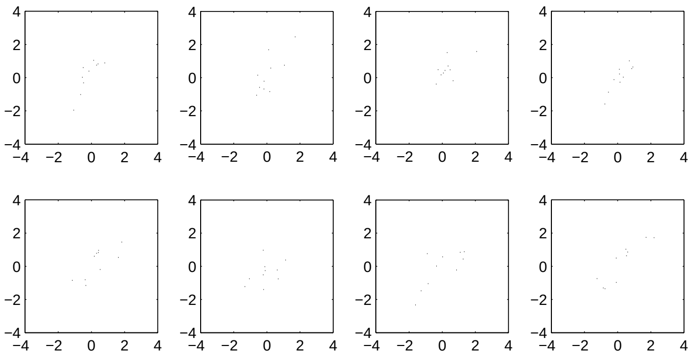

Figure 11: Original sources are on the x-axis of each scatter plot and the sources estimated by a linear ICA are on the y-axis. Signal to noise ratio is 0.7 dB

Several different numbers of hidden neurons were tested in order to optimise the structure of the network, but the number of sources was assumed to be known. This assumption is reasonable since it is possible to optimise the number of sources simply by minimising the cost function as the experiments with the Gaussian sources show. The network which minimised the cost function turned out to have 50 hidden neurons. The number of Gaussians in each of the mixtures modelling the distribution of each source was chosen to be three and no attempt was made to optimise this.

The results are depicted in Figs. 11, 12 and 13. Each figure shows eight scatter plots, each of which corresponds to one of the eight sources. The original source which was used for generating the data is on the x-axis and the estimated source in on the y-axis of each plot. Each point corresponds to one data vector. The upper plots of each figure correspond to the super-Gaussian and the lower plots to the sub-Gaussian sources. Optimal result would be a straight line which would mean that the estimated values of the sources coincide with the true values.

Figure 11 shows the result of a linear FastICA algorithm [4]. The linear ICA is able to retrieve the original sources with 0.7 dB signal to noise ratio. In practise a linear method could not deduce the number of sources and the result would be even worse. The poor signal to noise ratio shows that the data really lies in a nonlinear subspace.

Figure 12 depicts the results after 2000 iterations with nonlinear factor analysis followed by a rotation with a linear FastICA. Now the signal to noise ratio is 13.2 dB and the sources have clearly been retrieved. Figure 13 shows the final result after another 5500 iterations with nonlinear independent factor analysis algorithm. The signal to noise ratio has further improved to 17.3 dB.

It would also be possible to avoid using the nonlinear independent factor analysis algorithm by running first 7500 iterations with linear factor analysis algorithm and then applying the linear ICA. The disadvantage would be that the cost function would not take into account the non-Gaussianity. The signal to noise ratio after 7500 iterations with linear factor analysis algorithm followed by the linear ICA was 14.9 dB, which shows that taking

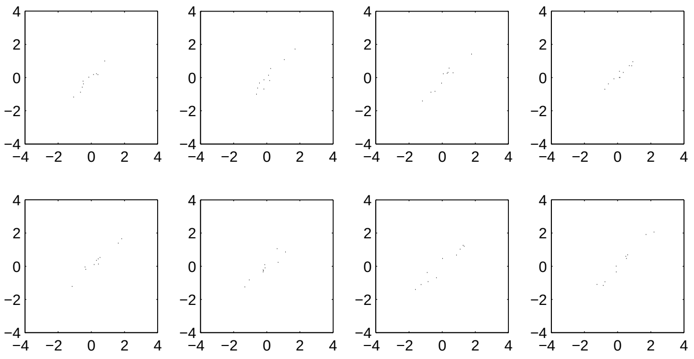

Figure 12: Scatter plots of the sources after 2000 iterations of nonlinear factor analysis followed by a rotation with a linear ICA. Signal to noise ratio is 13.2 dB

the non-Gaussianity into account during estimation of the nonlinear mapping helps the nonlinear independent factor analysis algorithm to find better estimates for the sources.

5.4 Process Data

This data set consists of 30 time series of length 2480 measured from different sensors from an industrial pulp process. An expert has preprocessed the signals by roughly compensating for time lags of the process which originate from the finite speed of pulp flow through the process.

In order to get an idea of the dimensionality of the data, linear factor analysis was applied to the data. The result is shown in Fig. 14. The same figure shows also the results with nonlinear factor analysis. It appears that the data is quite nonlinear since the nonlinear factor analysis is able to explain as much data with 10 components as the linear factor analysis with 21 components.

Several different numbers of hidden neurons and sources where tested with different random initialisations with nonlinear factor analysis and it turned out that the cost function was minimised for a network having 10 sources and 30 hidden neurons. The same network was chosen for nonlinear independent factor analysis, i.e., after 2000 iterations with linear factor analysis the sources were rotated with FastICA and each source was modelled with a mixture of three Gaussian distributions. The resulting sources are shown in Fig. 15.

Figure 16 shows the 30 original time series of the data set, one time series per plot, and in the same plots below the original time series are the reconstructions made by the network, i.e., the posterior means of the output of the network when the inputs were the estimated sources shown in Fig. 15. The original signals shown great variability but the reconstructions are strikingly accurate. In some cases it even seems that the reconstruction is less noisy than the original signal. This is somewhat surprising since the time dependencies in the signal were not included in the model. The observation vectors could be arbitrarily shuffled and the model would still give the same result.

Initial studies are pointing to the direction that the estimated source signals can have

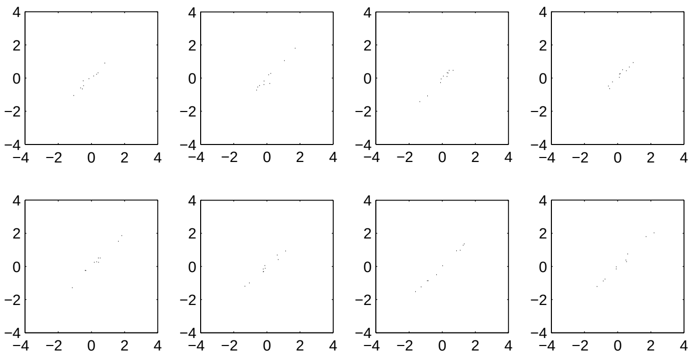

Figure 13: The network in Fig. 12 has been further trained for 5500 iterations with nonlinear independent factor analysis. Signal to noise ratio is 17.3 dB

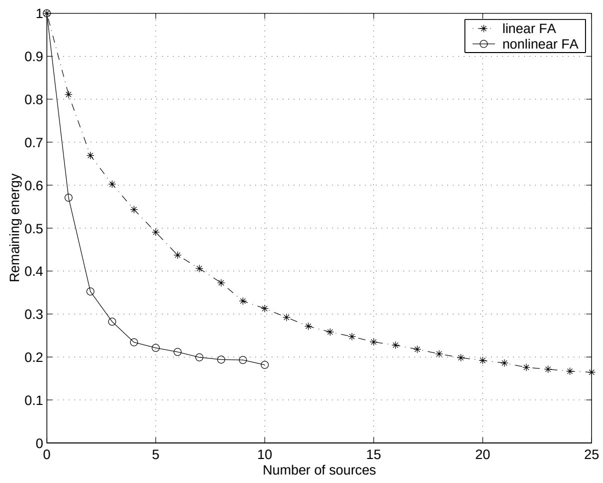

Figure 14: The graph shows the remaining energy in the process data as a function of the number of extracted components in linear and nonlinear factor analysis

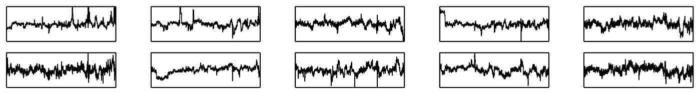

Figure 15: The ten estimated sources from the industrial pulp process. Time increases from left to right

meaningful physical interpretations. The results are encouraging but further studies are needed to verify the interpretations of the signals.

6 Comparison with Existing Methods

The idea of representing the data with a nonlinear coordinate system is by no means new and several algorithms for learning the coordinate system have been proposed.

6.1 SOM and GTM

Self-organising maps (SOM) [5] and generative topographic mapping (GTM) [2] define a nonlinear coordinate system by stating the coordinates of lattice points called model vectors. The methods are in wide use, particularly the computationally efficient SOM, but the dimension of the latent space is normally quite small. Two-dimensional latent space is the most typical one because it allows an easy visualisation for human users.

The disadvantage of SOM and GTM is that the number of parameters required to describe the mapping from latent variables to observations grows exponentially with the dimension of the latent space. Our main motivation for using MLP network as the nonlinear mapping is that its parametrisation scales linearly with the dimension of the latent space. In this respect the mapping of MLP network is much closer to a linear mapping which has been proven to be applicable for very high dimensional latent spaces. SOM and GTM would probably be better models for the helical data set in Sect. 5.2, but the rest of the experiments have latent spaces whose dimensions are so large that SOM or GTM models would need very many parameters.

6.2 Auto-Associative MLPs

Auto-associative MLP networks have been used for learning similar mappings as we have done. Both the generative model and its inversion are learned simultaneously, but separately without utilising the fact that the models are connected. This means that the learning is much slower than in this case where the inversion is defined as a gradient descent process.

Much of the work with auto-associative MLPs uses point estimates for weights and sources. As argued in the beginning of the chapter, it is then impossible to reliably choose the structure of the model and problems with over- or underlearning may be severe. Hochreiter and Schmidhuber have used and MDL based method which does estimate the distribution of the weights but has no model for the sources [3]. It is then impossible to measure the description length of the sources.

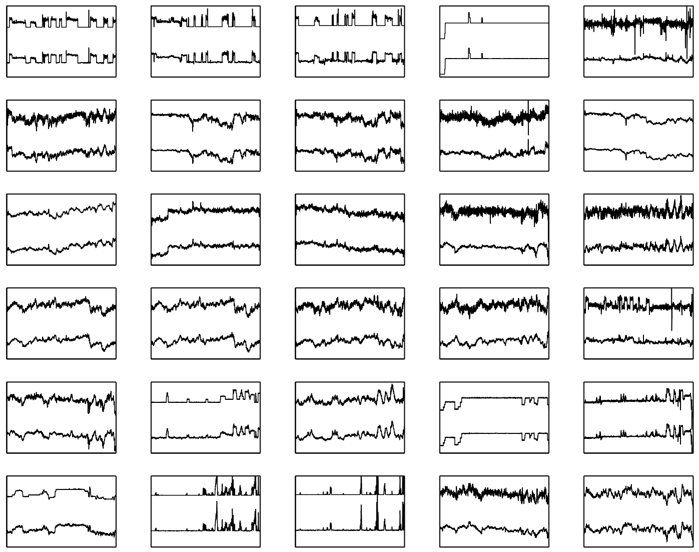

Figure 16: The 30 original time series are shown on each plot on top of the reconstruction made from the sources shown in Fig. 15

6.3 Generative Learning with MLPs

MacKay and Gibbs briefly report using stochastic approximation to learn a generative MLP network which they called a density network because the model defines a density of the observations [6]. Although the results are encouraging, they do not prove the advantages of the method over SOM or GTM because the model is very simple; noise level is not estimated from the observations and the latent space had only two dimensions. The computational complexity of the method is significantly greater than in the parametric approximation of the posterior presented here, but it might be possible to combine the methods by finding initial approximation of the posterior probability with parametric approximation and then refining it with more elaborate stochastic approximation.

In [7], a generative MLP network was optimised by gradient based learning. The cost function was reconstruction error of the data and a point estimate was used for all the unknown variables. As argued in Sect. 2, this means that it is not possible to optimise model structure and the method is prone to overfitting.

7 Discussion

7.1 Validity of the Approximations

The posterior pdf of all the unknown variables was approximated with a Gaussian density with diagonal covariance, which means that the variables were assumed independent given the observations. The Gaussianity assumption is not severe since the parametrisation is chosen so as to make the posterior close to Gaussian. If the hidden neurons were linear, the posterior of the sources, weights and biases would, in fact, be exactly Gaussian. Gaussian approximation therefore penalises strong nonlinearities to some extent.

The posterior independence seems to be the most unrealistic assumption. It is probable that a change in one of the weights can be compensated by changing the values of other weights and sources, which means that they have posterior dependencies.

In general, the cost function tries to make the approximation of the posterior more accurate, which means that during learning the posterior will also try to be more independent. In PCA, the mapping has a degeneracy which will be used by the algorithm to do exactly this. In linear PCA the mapping is such that the sources are independent a posteriori. In the nonlinear factor analysis, the dependencies of the sources are different in different parts of the latent space and it would be reasonable to model these dependencies. Computational load would not increase significantly since the Jacobian matrix computed in the algorithm can be used also for estimating the posterior interdependencies of the sources. For the sake of simplicity, the derivations were not included here.

It should be possible to do the same for the nonlinear independent factor analysis, but it would probably be necessary to assume the different Gaussians of each source to be independent. Otherwise the posterior approximation of $Q(\mathrm{M}|\mathrm{X})$ would be computationally too intensive.

The other approximation was done when approximating the nonlinearities of the hidden neurons by Taylor's series expansions. For small variances this is valid and it is therefore good to check that the variances of the inputs for the hidden neurons are not outside the range where the approximation is valid. In the computation of the gradients, some terms were neglected to discourage the network from adapting itself to areas of parameter space where the approximation is inaccurate. Experiments have proven that this seems to be working. For the network which minimised the cost function in Fig. 8, for instance, the maximum variance of the input for a hidden neurons was 0.06. Even this maximum value is safely below the values where the approximation could become too inaccurate.

7.2 Initial Inversion by Auxiliary MLP

During learning, both the sources and the mapping of the network evolve together. When the network is presented new data, it is necessary to find the estimates of the sources corresponding to the new data. This can be difficult using similar update process as was used in learning because it is possible that during learning the network develops local minima which make later inversion difficult.

Experiments have shown that it is possible to learn an auxiliary MLP network which will estimate the mapping from observations to sources and can thus be used to initialise the sources given new data. The resulting system with two MLP networks resembles auto-associative MLP network. As was argued before, learning only the generative model is faster than learning a deeper auto-associative MLP with both the generative model and its inverse. Initial experiments have also shown that updates of the sources after the initialisation with the auxiliary MLP network lead to better estimates of the sources.

7.3 Future Directions

In principle, both the nonlinear factor analysis and independent factor analysis can model any time-independent distribution of the observations. MLP networks are universal approximators for mappings and mixture-of-Gaussians for densities. This does not mean, however, that the models described here would be optimal for any time-independent data sets, but the Bayesian methods which were used in the derivation of the algorithms allow easy extensions to more complicated models. It is also easy to use Bayesian model comparison to decide with model is most suited for the data set at hand.

An important extension would be the modelling of dependencies between consecutive sources $\vec{s}(t)$ and $\vec{s}(t+1)$ because many natural data sets are time series. For instance both the speech and process data sets used in the experiments clearly have strong time-dependencies.

In the Bayesian framework, treatment of missing values is simple which opens up interesting possibilities for the nonlinear models described here. A typical pattern recognition task can often be divided in unsupervised feature extraction and supervised recognition phases. Using the proposed method, the MLP network can be used for both phases. The data set for the unsupervised feature extraction would have only the raw data and the classifications would be missing. The data set for supervised phase would include both the raw data and the desired outputs and the network. From the point of view of the method presented here, there is no need to make a clear distinction between unsupervised and supervised learning phases as any data vectors can have any combination of missing values. The network will model the joint distribution of all the observations and it is not necessary to specify which of the variables will be the classifications and which are the raw data.

Acknowledgements

The authors wish to thank M.Sc. Xavier Giannakopoulos for his help with the simulations with the nonlinear independent factor analysis and M.Sc. Esa Alhoniemi for his help with the pulp process data.

References

[1] Hagai Attias. Independent factor analysis. *Neural Computation*, 11(4):803–851, 1999.

[2] Christopher M. Bishop, Markus Svensén, and Christopher K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998.

[3] Sepp Hochreiter and Jürgen Schmidhuber. LOCOCODE performs nonlinear ICA without knowing the number of sources. In *Proceedings of the ICA'99*, pages 149–154, Aussois, France, 1999.

[4] A. Hyvärinen and E. Oja. A fast fixed-point algorithm for independent component analysis. *Neural Computation*, 9(7):1483–1492, 1997.

[5] Teuvo Kohonen. *Self-Organizing Maps*. Springer, 1995.

[6] David J. C. MacKay and Mark N. Gibbs. Density networks. In Jim Kay, editor, *Proceedings of Society for General Microbiology Edinburgh meeting*, 1997.

[7] Jong-Hoon Oh and H. Sebastian Seung. Learning generative models with the up-propagation algorithm. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 605–611. MIT Press, 1998.

NONLINEAR INDEPENDENT COMPONENT ANALYSIS USING ENSEMBLE LEARNING: THEORY

Harri Valpola

In *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, (Helsinki, Finland, June 19–22), pp. 251–256, 2000.

Abstract

A nonlinear version of independent component analysis is presented. The mapping from sources to observations is modelled by a multi-layer perceptron network and the distributions of sources are modelled by mixtures of Gaussians. The posterior probability of all the unknown parameters is estimated by ensemble learning. In this paper, we present the theory of the method, and in a companion paper experimental results.

1 Introduction

This paper presents a nonlinear independent component analysis (ICA) algorithm. In much of the ICA research the mapping from the sources $\mathbf{s}(t)$ to the observations $\mathbf{x}(t)$ has been assumed to be linear, but here we consider a more general model where the observations are assumed to be generated by a nonlinear mapping $\mathbf{f}$ from the sources as shown in (1).

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t) \tag{1}$$

The observations are assumed to be corrupted by Gaussian noise $\mathbf{n}(t)$. The sources are assumed to be generated by an i.i.d. process and the distribution of each of the sources is modelled by a mixture of Gaussians (MOG) and the nonlinear mapping $\mathbf{f}$ is modelled by a multi-layer perceptron (MLP) network.

It is well known that the problem of determining the nonlinearity $\mathbf{f}$ is indeterminate, that is, there exists an infinite number of different nonlinearities which can produce observations with the same distribution from some independent sources [2]. The Bayesian approach does not suffer from the indeterminacy because a posterior probability can be assigned to all the nonlinear models. Here we use ensemble learning which is a computationally efficient approximation to the full Bayesian treatment [4]. Although each model will have some probability of being responsible for generating the data, typically almost all probability is concentrated on a very small subset of models. It suffices to approximate this high probability region of the posterior probability of the models. In ensemble learning, a parametric approximation is fitted to the posterior probability.

The model structure is presented in Sect. 2. The cost function and details about how to efficiently evaluate it are discussed in Sect. 3. Finally, the learning algorithm is introduced in Sect. 4. This paper is accompanied by [5] which demonstrates the feasibility of the method in simulations with artificial and natural data sets and discusses other existing nonlinear ICA algorithms as well as the limitations and possible extensions of the algorithm described here.

2 Model Structure

The nonlinear mapping $\mathbf{f}$ is modelled by a multi-layer perceptron (MLP) network having two layers.

$$\mathbf{f}(\mathbf{s}(t)) = \mathrm{B}\mathbf{g}(\mathrm{A}\mathbf{s}(t) + \mathbf{a}) + \mathbf{b} \tag{2}$$

The activation function for each of the nonlinear hidden neurons is the hyperbolic tangent, that is, $g(y) = \tanh(y)$. In addition to the weight matrices A and B, both the hidden neurons and the linear output neurons have biases, denoted by $\mathbf{a}$ and $\mathbf{b}$, respectively.

In order to apply the Bayesian approach, each unknown variable in the network is assigned a probability density function (pdf). We apply the usual hierarchical definition of priors. For many parameters, for instance the biases $\mathbf{a}$, it is difficult to assign a prior distribution but we can utilise the fact that each bias occurs in a similar role in the network by assuming that the distribution for each element of vector $\mathbf{a}$ has the same, albeit unknown distribution which is then modelled by a parametric distribution. These new parameters need to be assigned a prior also, but there are far fewer of them.

The noise $\mathbf{n}(t)$ is assumed to be independent and Gaussian with a zero mean. The variance can be different on different channels, and hence the algorithm can be more accurately be called nonlinear independent factor analysis. Given $\mathbf{s}(t)$, the variance of $\mathbf{x}(t)$ is due to the noise. Therefore $\mathbf{x}(t)$ has the same distribution as the noise except with the mean $\mathbf{f}(\mathbf{s}(t))$.

The distribution of each of the sources is modelled by a mixture of Gaussians. We can think that for each source $s_i(t)$ there is a discrete process which produces a sequence $M_i(t)$ of indices which tell from which Gaussian each $s_i(t)$ is originated. Each Gaussian has its own mean and variance and the probability of different indices is modelled by a soft-max distribution.

The model is defined by the following set of distributions:

$$\begin{aligned}
\mathbf{x}(t) &\sim N(\mathbf{f}(\mathbf{s}(t)), \exp(2\mathbf{v}_n)) && (3)\\
P(M_i(t) = l) &= \exp(c_{il}) / \sum_{l'} \exp(c_{il'}) && (4)\\
s_i(t) &\sim N(m_{sil}, \exp(2v_{sil})) && (5)\\
\mathrm{A} &\sim N(0, 1) && (6)\\
\mathrm{B} &\sim N(0, \exp(2\mathbf{v}_B)) && (7)\\
\mathbf{a} &\sim N(m_a, \exp(2v_a)) && (8)\\
\mathbf{b} &\sim N(m_b, \exp(2v_b)) && (9)\\
\mathbf{v}_n &\sim N(m_{v_n}, \exp(2v_{v_n})) && (10)\\
\mathbf{c} &\sim N(0, \exp(2v_c)) && (11)\\
\mathbf{m}_s &\sim N(0, \exp(2v_{m_s})) && (12)\\
\mathbf{v}_s &\sim N(m_{v_s}, \exp(2v_{v_s})) && (13)\\
\mathbf{v}_B &\sim N(m_{v_B}, \exp(2v_{v_B})) && (14)
\end{aligned}$$

The prior distributions of m_a, v_a, m_b, v_b and the eight hyperparameters $m_{v_n}, \ldots, v_{v_B}$ are assumed to be Gaussian with zero mean and standard deviation 100, that is, the priors are assumed to be very flat.

The parametrisation of all the distributions is chosen such that the resulting parameters have a roughly Gaussian posterior distribution. This is because the posterior will be modelled by a Gaussian distribution. For example, the variance of the Gaussian distributions is parametrised on a logarithmic scale.

Model indeterminacies are handled by restricting some of the distributions. There is a scaling indeterminacy between the matrix A and the sources, for instance. This is taken care of by setting the variance of A to unity instead of parametrising and estimating it. For the second layer matrix B there is no such indeterminacy. The variance of each column of the matrix is $\exp(2v_{Bj})$. The network can effectively prune out some of hidden neurons by setting the outgoing weights of the hidden neurons to zero, and this is easier if the variance of the corresponding columns of B can be given small values.

3 Cost Function

The goal is to estimate the posterior pdf of all the unknown variables of the model. This is done by ensemble learning which amounts to fitting a simple, parametric approximation to the actual posterior pdf [4]. The cost function C is the misfit between the approximation and the actual posterior and is measured by the Kullback-Leibler information which is sensitive to the probability mass of densities. This is the most important advantage over maximum a posteriori (MAP) estimation which is computationally less expensive but is sensitive to probability density, not mass. This is why MAP estimation suffers from overfitting, which would be a serious problem since there are so many estimated variables, while ensemble learning is able avoid it.

For the time being, let us denote the set of all observations vectors $\mathbf{x}(t)$ by X and denote all the other parameters by a vector θ. The actual posterior pdf is thus $p(\boldsymbol{\theta}|X) = p(X,\boldsymbol{\theta})/p(X)$. The joint pdf $p(X,\boldsymbol{\theta})$ is obtained from the definition of the model in (3)–(14) and $p(X)$ is a normalising factor which does not depend on the unknown variables.

Let us denote the approximation of the posterior pdf by $q(\boldsymbol{\theta})$. In order for the cost function to be computable in practice, a simple factorial form needs to be chosen for the approximation $q(\boldsymbol{\theta})$. The maximally factorial form would be

$$q(\boldsymbol{\theta}) = \prod_i q(\theta_i)\,. \tag{15}$$

Notice that we have used the usual notation with probability density functions where q with different arguments are taken to be different functions.

The assumption of factorial $q(\boldsymbol{\theta})$ is equivalent to assuming the unknown variables independent given the observations. This is not true, of course, but we have to make this approximation in order to obtain a practical algorithm. The only exception to this maximally factorial form is that the index $M_i(t)$ of the Gaussian and the corresponding source $s_i(t)$ are allowed to have posterior dependency, that is, the terms $q(M_i(t), s_i(t))$ are not further factorised.

The approximation $q(\theta_i)$ should be chosen so that it fits the actual posterior as closely as possible. This is accomplished by choosing $q(\theta_i)$ to be Gaussian for other variables than sources and for sources choosing $q(M_i(t), s_i(t)) = Q(M_i(t))q(s_i(t)|M_i(t))$, where $q(s_i(t)|M_i(t))$ is Gaussian.

Let us denote the mean and variance of $q(\theta_i)$ by $\bar{\theta}_i$ and $\tilde{\theta}_i$, respectively. The result of learning is then an estimate of $\bar{\boldsymbol{\theta}}$ and $\tilde{\boldsymbol{\theta}}$ which tell the posterior mean and variance of all the unknown variables.

The term $p(X)$ is constant with respect to the unknown parameters. Instead of the pure Kullback-Leibler information $K(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|X))$ it is therefore possible to use the following cost function:

$$\begin{aligned} C(\bar{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}}) &= K(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|X)) - \ln p(X) = \\ & \int q(\boldsymbol{\theta}) \ln \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|X)} d\boldsymbol{\theta} - \ln p(X) = \\ & \int q(\boldsymbol{\theta}) \ln \frac{q(\boldsymbol{\theta})}{p(X,\boldsymbol{\theta})} d\boldsymbol{\theta}\,. \end{aligned} \tag{16}$$

Notice that the variables $M_i(t)$ are discrete and those terms are summed over, not integrated over, in the Kullback-Leibler information. For simplicity this is omitted from (16).

Due to simple factorial forms of $q(\boldsymbol{\theta})$ and $p(X,\boldsymbol{\theta})$ the cost function splits into simple terms which are easy to compute. Consequently, it is also easy to differentiate the cost function with respect to $\bar{\boldsymbol{\theta}}$ and $\tilde{\boldsymbol{\theta}}$ and use the derivatives for constructing the learning algorithm.

3.1 Terms of the Cost Function

Almost all the distributions appearing in our model are assumed to be Gaussians, and consequently almost all the terms appearing in the cost function are expectations of logarithms of Gaussian distributions. We shall use the second layer biases $\mathbf{a}$ as an example. For each element a_i, there is one term in $q(\boldsymbol{\theta})$ and $p(X,\boldsymbol{\theta})$, namely the terms $q(a_i)$ and $p(a_i|m_a, v_a)$. The cost function therefore includes terms $\int q(\boldsymbol{\theta}) \ln q(a_i) d\boldsymbol{\theta}$ and $-\int q(\boldsymbol{\theta}) \ln p(a_i|m_a, v_a) d\boldsymbol{\theta}$. In the first expectation the terms $q(a_i)$ only depends on a_i which means that we can integrate over the other variables and we have

$$\int q(\boldsymbol{\theta}) \ln q(a_i) d\boldsymbol{\theta} = \int q(a_i) \ln q(a_i) da_i\,. \tag{17}$$

The same happens for the other integral:

$$\begin{aligned} -\int q(\boldsymbol{\theta}) \ln p(a_i|m_a, v_a) d\boldsymbol{\theta} = \\ -\int q(a_i)q(m_a)q(v_a) \ln p(a_i|m_a, v_a) da_i dm_a dv_a\,. \end{aligned} \tag{18}$$

Recall that $q(a_i)$ is Gaussian with mean $\bar{a}_i$ and variance $\tilde{a}_i$. This means that the integral in (17) yields simply

$$\int q(a_i) \ln q(a_i) da_i = -\frac{1}{2} \ln 2\pi e \tilde{a}_i\,. \tag{19}$$

The integral in (18) also fairly easy and it can be shown that the result is

$$\begin{aligned} -\int q(a_i)q(m_a)q(v_a) \ln p(a_i|m_a, v_a) da_i dm_a dv_a = \\ \frac{1}{2}[(\bar{a}_i - \bar{m}_a)^2 + \tilde{a}_i + \tilde{m}_a] \exp(2\tilde{v}_a - 2\bar{v}_a) + \bar{v}_a + \frac{1}{2}\ln 2\pi\,. \end{aligned} \tag{20}$$

Again the result is based on the fact that $q(a_i)$, $q(m_a)$ and $q(v_a)$ are Gaussian with means $\bar{a}_i$, $\bar{m}_a$, $\bar{v}_a$ and variances $\tilde{a}_i$, $\tilde{m}_a$, $\tilde{v}_a$, respectively.

The following terms are the only ones whose expectations in (16) give different results than (19) or (20): $q(M_i(t), s_i(t))$, $p(M_i(t)|\mathbf{c}_i)$, $p(s_i(t)|M_i(t), \mathbf{m}_{si}, \mathbf{v}_{si})$ and $p(x_k(t)|\boldsymbol{\theta})$.

The index $M_i(t)$ is discrete and therefore we have a summation instead of integration in the cost function. Let us denote $\dot{s}_{il}(t) = Q(M_i(t) = l)$ and denote the mean and variance of the Gaussian $q(s_i(t)|M_i(t) = l)$ by $\bar{s}_{il}(t)$ and $\tilde{s}_{il}(t)$. Then the expectations of $\ln q(M_i(t), s_i(t))$ in (16) are given by

$$\sum_l \int q(\boldsymbol{\theta}) \ln q(M_i(t) = l, s_i(t)) d\boldsymbol{\theta} = \sum_l Q(M_i(t) = l)[\ln Q(M_i(t) = l) + \int q(s_i(t)|M_i(t) = l) \ln q(s_i(t)|M_i(t) = l) ds_i(t)] = \sum_l \dot{s}_{il}(t)[\ln \dot{s}_{il}(t) - \frac{1}{2} \ln 2\pi e \tilde{s}_{il}(t)] . \quad (21)$$

For the expectation of $-\ln p(M_i(t)|\mathbf{c}_i)$ we shall first evaluate the following integral:

$$-\int q(\mathbf{c}_i) \ln p(M_i(t) = l|\mathbf{c}_i) d\mathbf{c}_i = -\int q(\mathbf{c}_i)[c_{il} - \ln \sum_{l'} \exp(c_{il'})] d\mathbf{c}_i = -\bar{c}_{il} + \int q(\mathbf{c}_i) \ln \sum_{l'} \exp(c_{il'}) d\mathbf{c}_i \quad (22)$$

The resulting integral can be approximated by applying a second order Taylor's series expansion of $\ln \sum_{l'} \exp(c_{il'})$ with respect to $c_{il'}$ around the posterior mean $\bar{c}_{il'}$. This yields the following approximation for the integral:

$$-\int q(\mathbf{c}_i) \ln p(M_i(t) = l|\mathbf{c}_i) d\mathbf{c}_i \approx -\bar{c}_{il} + \frac{1}{2} \sum_{l'} \phi_{il'}(1 - \phi_{il'}) \tilde{c}_{il'} , \quad (23)$$

where $\phi_{il} = \exp(\bar{c}_{il}) / \sum_{l'} \exp(\bar{c}_{il'})$. Now we see that the expectation of $-\ln p(M_i(t)|\mathbf{c}_i)$ is

$$-\sum_j \dot{s}_{il} \int q(\mathbf{c}_i) \ln p(M_i(t)|\mathbf{c}_i) d\mathbf{c}_i \approx -\sum_l \dot{s}_{il} \bar{c}_{il} + \frac{1}{2} \sum_{l'} \phi_{il'}(1 - \phi_{il'}) \tilde{c}_{il'} . \quad (24)$$

Since both $q(s_i(t)|M_i(t))$ and $p(s_i(t)|M_i(t), \mathbf{m}_{si}, \mathbf{v}_{si})$ are Gaussian, the terms

$-\ln p(s_i(t)|M_i(t), \mathbf{m}_{si}, \mathbf{v}_{si})$ have expectations which are similar to (20):

$$-\sum_l \dot{s}_{il} \int q(s_i(t)|M_i(t) = l) q(m_{sil}) q(v_{sil}) \ln p(s_i(t)|M_i(t) = l, m_{sil}, v_{sil}) ds_i(t) dm_{sil} dv_{sil} , \quad (25)$$

which equals to the sum of terms

$$\frac{1}{2}[(\bar{s}_{il}(t) - \bar{m}_{sil})^2 + \tilde{s}_{il}(t) + \tilde{m}_{sil}] \exp(2\tilde{v}_{sil} - 2\bar{v}_{sil}) + \bar{v}_{sil} + \frac{1}{2} \ln 2\pi \quad (26)$$

weighted by $\dot{s}_{il}$.

The observations $x_k(t)$ are known — unless there are missing values — which means that there are no terms of the form $\ln q(x_k(t))$. The expectations of $-\ln p(x_k(t)|\boldsymbol{\theta})$ are the most difficult terms in the cost function. If the posterior mean and variance of the function $f_k(\mathbf{s}(t))$ are known — let us denote them by $\bar{f}_k(t)$ and $\tilde{f}_k(t)$ for short — then the expectation has a form similar to (20):

$$\frac{1}{2}[(x_k(t) - \bar{f}_k(t))^2 + \tilde{f}_k(t)] \exp(2\tilde{v}_{nk} - 2\bar{v}_{nk}) + \bar{v}_{nk} + \frac{1}{2} \ln 2\pi . \quad (27)$$

3.2 Posterior Mean and Variance of $\mathbf{f}(\mathbf{s}(t))$

This section describes how to compute the posterior mean and variance of the outputs $f_k(\mathbf{s}(t))$ of the MLP network. Ordinarily the inputs, weights and biases of an MLP network have fixed values. Here the inputs $\mathbf{s}(t)$, weights A, B and the biases $\mathbf{a}$, $\mathbf{b}$ have posterior distributions which means that we also have a posterior distribution of the outputs. One way to evaluate the posterior mean and variance is to propagate distributions instead of fixed values through the network. Whole distributions would be quite tricky to deal with, and therefore we are going to characterise the distributions by their mean and variance only.

The sources have mixture-of-Gaussians distributions for which it is easy to compute the mean and variance:

$$\bar{s}_i(t) = \sum_l \dot{s}_{il}(t) \bar{s}_{il}(t) \quad (28)$$

$$\tilde{s}_i(t) = \sum_l \dot{s}_{il}(t)[\tilde{s}_{il}(t) + (\bar{s}_{il}(t) - \bar{s}_i(t))^2] . \quad (29)$$

Then the sources are multiplied with the first layer weight matrix A and the bias $\mathbf{a}$ is added. Let us denote the result by $y_j(t) = a_j + \sum_i A_{ji} s_i(t)$. Since the sources, weights and biases are all mutually independent a posteriori, the following equations hold:

$$\bar{y}_j(t) = \bar{a}_j + \sum_i \bar{A}_{ji} \bar{s}_i(t) \quad (30)$$

$$\tilde{y}_j(t) = \tilde{a}_j + \sum_i \bar{A}_{ji}^2 \tilde{s}_i(t) + \tilde{A}_{ji}[\bar{s}_i^2(t) + \tilde{s}_i(t)] . \quad (31)$$

Equation (31) follows from the identity

$$\mathrm{var}(\alpha) = \langle \alpha^2 \rangle - \langle \alpha \rangle^2 \,. \tag{32}$$

For computing the posterior mean of the output $g_j(y_j(t))$ of a hidden neuron, we shall utilise the second order Taylor's series expansion of g_j around the posterior mean $\bar{y}_j(t)$ of its input. This means that we approximate

$$\begin{aligned} g_j(y_j(t)) \quad &\approx \quad g_j(\bar{y}_j(t)) + (y_j(t) - \bar{y}_j(t))g_i'(\bar{y}_j(t)) + \\ &\quad\ \frac{1}{2}(y_j(t) - \bar{y}_j(t))^2 g_i''(\bar{y}_j(t)) \,. \end{aligned} \tag{33}$$

Since the posterior mean of $y_j(t)$ is by definition $\bar{y}_j(t)$, the second term vanishes when evaluating the posterior mean, while the posterior mean of $(y_j(t) - \bar{y}_j(t))^2$ is by definition the posterior variance $\tilde{y}_j(t)$. We thus have

$$\bar{g}_j(y_j(t)) \approx g_j(\bar{y}_j(t)) + \frac{1}{2}\tilde{y}_j(t) g_j''(\bar{y}_j(t)) \,. \tag{34}$$

The second order expansion was chosen because those are the terms whose posterior mean can be expressed in terms of posterior mean and variance of the input. Higher order terms would have required higher order cumulants of the input, which would have increased the computational complexity with little extra benefit.

For the posterior variance of $g_j(y_j(t))$ the second order expansion would result in terms which need higher than second order knowledge about the inputs. Therefore we shall use the first order Taylor's series expansion which then yields the following approximation for the posterior variance of $g_j(y_j(t))$:

$$\tilde{g}_j(y_j(t)) \approx [g_j'(\bar{y}_j(t))]^2 \tilde{y}_j(t) \,. \tag{35}$$

The next step is to compute the mean and variance of the output after the second layer mapping. The outputs are given by $f_k(t) = b_k + \sum_j B_{kj} g_j(t)$. The equation for the posterior mean $\bar{f}_k(t)$ is similar to (30):

$$\bar{f}_k(t) = \bar{b}_k + \sum_j \bar{B}_{kj} \bar{g}_j(t) \,. \tag{36}$$

The equation for the posterior variance $\tilde{f}_k(t)$ is more complicated than (31), however, since $s_i(t)$ are independent a posterior but $g_j(t)$ are not. This is because each $s_i(t)$ affects several — potentially all — $g_j(t)$. In other words, each $s_i(t)$ affects each $f_k(t)$ through several paths which interfere. This interference needs to be taken into account when computing the posterior variance of $f_k(t)$.

We shall use a first order approximation of the mapping $\mathbf{f}(\mathbf{s}(t))$ for measuring the interference. This is consistent with the first order approximation of the nonlinearities g_j and yields the following equation for the posterior variance of $f_k(t)$:

$$\begin{aligned} \tilde{f}_k(t) \quad &\approx \quad \sum_i \left(\frac{\partial f_k(t)}{\partial s_i(t)} \right)^2 \tilde{s}_i(t) + \tilde{b}_k + \\ &\quad\ \sum_j \bar{B}_{kj}^2 \tilde{g}_j^*(t) + \tilde{B}_{jk}[\bar{g}_j^2(t) + \tilde{g}_j(t)] \,, \end{aligned} \tag{37}$$

where the posterior means of the partial derivatives are obtained by the chain rule

$$\begin{aligned} \frac{\partial f_k(t)}{\partial s_i(t)} \quad &= \quad \sum_j \frac{\partial f_k(t)}{\partial g_j(t)} \frac{\partial g_j(t)}{\partial y_j(t)} \frac{\partial y_j(t)}{\partial s_i(t)} = \\ &\quad\ \sum_j \bar{B}_{kj} g_j'(\bar{y}_j(t)) \bar{A}_{ji} \end{aligned} \tag{38}$$

and $\tilde{g}_j^*(t)$ denotes the posterior variance of $g_j(t)$ without the contribution from the sources. It can be computed as follows:

$$\tilde{y}_j^*(t) \quad = \quad \tilde{a}_j + \sum_i \tilde{A}_{ji}[\bar{s}_i^2(t) + \tilde{s}_i(t)] \tag{39}$$

$$\tilde{g}_j^*(t) \quad \approx \quad [g_j'(\bar{y}_j(t))]^2 \tilde{y}_j^*(t) \,. \tag{40}$$

Notice that $\tilde{s}_i(t)$ appears in (39) and $\tilde{g}_j(t)$ appears in (37). These terms do not contribute to interference, however, because they are the parts which are randomised by multiplication with A_{ji} or B_{kj} and randomising the phase destroys the interference, to use an analogy from physics.

4 Update Rules

In the previous section we derived all the equations needed for the computation of the cost function. Given the posterior means $\bar{\boldsymbol{\theta}}$ and variances $\tilde{\boldsymbol{\theta}}$ and discrete posterior probabilities $\dot{s}_{il}(t)$, we can compute the cost function which measures the quality of the approximation of the posterior pdf of the unknown variables. Any standard optimisation algorithm could be used for minimising the cost function, but it is sensible to utilise the particular form of the function. Due to lack of space, we shall only outline the update rules but a more detailed description can be found in [3].

Let us denote $C = C_q + C_p$, where C_q is the part originating from the expectation of $\ln q(\boldsymbol{\theta})$ and C_p is the part originating from expectation of $-\ln p(X, \boldsymbol{\theta})$. We shall see how it is possible to derive efficient fixed point algorithms for $\bar{\theta}$ and $\tilde{\theta}$ assuming that we have computed the gradients of C_p with respect to the current estimates of $\bar{\theta}$ and $\tilde{\theta}$.

Since C_q has a term $-1/2 \ln 2\pi e \tilde{\theta}$ for each $\tilde{\theta}$ whose posterior is approximated by Gaussian $q(\theta)$, solving for $\partial C / \partial \tilde{\theta} = 0$ yields an update rule for $\tilde{\theta}$:

$$0 = \frac{\partial C_p}{\partial \tilde{\theta}} + \frac{\partial C_q}{\partial \tilde{\theta}} = \frac{\partial C_p}{\partial \tilde{\theta}} - \frac{1}{2\tilde{\theta}} \ \Rightarrow \ \tilde{\theta} = \frac{1}{2 \frac{\partial C_p}{\partial \tilde{\theta}}} \,. \tag{41}$$

Now suppose $\ln p(X, \boldsymbol{\theta})$ is roughly quadratic with respect to θ:

$$-\ln p(X, \boldsymbol{\theta}) \approx \alpha + (\theta - \theta_{\mathrm{opt}})^2 \beta \,. \tag{42}$$

Then C_p would be

$$C_p \approx \alpha + [(\bar{\theta} - \theta_{\mathrm{opt}})^2 + \tilde{\theta}]\beta \tag{43}$$

and hence the derivatives with respect to $\bar{\theta}$ and $\tilde{\theta}$ would be

$$\frac{\partial C_p}{\partial \bar{\theta}} \quad = \quad 2(\bar{\theta} - \theta_{\mathrm{opt}})\beta \tag{44}$$

$$\frac{\partial C_p}{\partial \tilde{\theta}} \quad = \quad \beta \,. \tag{45}$$

As C_q does not depend on $\bar{\theta}$, the optimal value for $\bar{\theta}$ is evidently θ_{opt} and solving for that we obtain an update rule for $\bar{\theta}$:

$$\bar{\theta}_{\text{new}} = \theta_{\text{opt}} = \bar{\theta}_{\text{old}} - \frac{\frac{\partial C_p}{\partial \bar{\theta}}}{2\frac{\partial C_p}{\partial \tilde{\theta}}} = \bar{\theta}_{\text{old}} - \frac{\partial C_p}{\partial \bar{\theta}} \tilde{\theta}\,. \tag{46}$$

Since this update rule makes a quadratic approximation of the cost function C, it can be viewed as Newton iteration which assumes that $\bar{\theta}$ is the only variable which changes because the quadratic approximation does not take into account the cross terms $\partial^2 C / \partial\theta_i \partial\theta_j$. In practice all the weights A and B, for instance, are adapted simultaneously and each weight affects the optimal value of the other weights. In [3] it is explained how it is possible to compensate for the error which results in the invalid assumption of independent adaptations.

4.1 Update Rules for Posterior Source Distributions

The posterior distributions of the sources are effectively approximated by mixtures of Gaussians which means that the above update rules are not directly applicable for them. The new values for discrete posterior probabilities of the source indices $\dot{s}_{il}(t)$ and the posterior means $\bar{s}_{il}(t)$ and variances $\tilde{s}_{il}(t)$ of the Gaussians corresponding to different source are most easily solved by making a quadratic approximation for C_f based on the derivatives $\partial C_f(t)/\partial \bar{s}_i(t)$ and $\partial C_f / \partial \tilde{s}_i(t)$, where C_f denotes the sum of terms of the form (27).

The usefulness of this approximation is based on the fact that the cost function tries to minimise the misfit between the approximation and the actual posterior. If we can solve the actual posterior and show that it can be described by our parametric approximation, we know that the cost function will be minimised by setting the parameters of the approximation to values corresponding to the actual posterior.

In this case, making a second order approximation for C_f is equivalent to approximating $p(X|s_i(t))$ by an unnormalised Gaussian distribution. Since the prior $p(s_i(t))$ is a mixture of Gaussians and the posterior will be given by $p(s_i(t)|X) = p(X|s_i(t))p(s_i(t))/p(X)$, we notice that the posterior is also a mixture of Gaussians since a Gaussian multiplied with an unnormalised Gaussian will produce another unnormalised Gaussian and the normalising factor $p(X)$ will then make sure that the posterior is a normalised mixture of Gaussians. From the resulting mixture of Gaussians one can then determine the values for $\dot{s}_{il}(t)$, $\bar{s}_{il}(t)$ and $\tilde{s}_{il}(t)$. In [1], a similar method without approximations was used for a linear model. Due to the linearity of the mapping, $\ln p(X|s_i(t))$ is quadratic and no approximations are needed.

4.2 Avoiding Problems Originating from Approximations

When constructing a learning algorithm which is based on approximations of the cost function, it is important to make sure that learning does not drive the network into areas of the parameter space where the approximations are no longer valid.

The approximations in (34) and (35) are based on a roughly quadratic or linear behaviour of the nonlinearities. This assumption is quite good if the posterior variance $\tilde{y}_j(t)$ of the inputs to the hidden neurons is not very large.

Since the approximations take into account only local behaviour of the nonlinearities g_j and MLP networks typically have multimodal posterior distributions, there must be areas of the parameter space where the second order derivative of the posterior probability with

respect to one of the parameters θ is positive. This means that $\partial C_p / \partial \tilde{\theta}$ is negative which in turn means that it appears that the cost function can be made arbitrarily small by letting $\tilde{\theta}$ grow.

It is easy to see that the problem is due to the local estimate of g since the logarithm of the posterior eventually has to go to negative infinity. The derivative $\partial C_p / \partial \tilde{\theta}$ will thus be positive for large $\tilde{\theta}$, but the local estimate of g_j fails to account for this.

In order to discourage the network from adapting itself into areas of parameter space where the problems might occur and to deal with the problem if it nevertheless occurred, the terms in (34) which have negative contribution to $\partial C_p / \partial \tilde{\theta}$ will be neglected in the computation of the gradients. As this can only make the estimate of $\tilde{\theta}$ in (41) smaller, this leads, in general, to increasing the accuracy of the approximations in (34) and (35).

5 Computational Complexity

Most of the computation in the forward phase is spent in (38). The computation of the gradients of (38) is also where most computation of the backward phase takes place. We have previously tried making the assumption that the outputs of the hidden neurons are independent a posteriori which then obviates the need of equation (38) because (37) can be replaced by an equation similar to (31). Simulations have shown that this assumption is too inaccurate. The computational complexity of this algorithm is proportional to $IJKT$, where I, J, K and T denote the source dimension, the number of hidden neurons, the number of outputs and the number of observation vectors. In a typical case $K > I$ which means that the computational complexity of the second layer dominates and the computational complexity is higher than in ordinary back-propagation by a factor I.

References

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

[2] A. Hyvärinen and P. Pajunen. Nonlinear independent component analysis: Existence and uniqueness results. *Neural Networks*, 12(2):209–219, February 1999.

[3] H. Lappalainen and A. Honkela. Bayesian nonlinear independent component analysis by multi-layer perceptrons. In M. Girolami, ed., *Advances in Independent Component Analysis*. Springer, Berlin, 2000. In Press.

[4] H. Lappalainen and J. W. Miskin. Ensemble learning. In M. Girolami, ed., *Advances in Independent Component Analysis*. Springer, Berlin, 2000. In Press.

[5] H. Valpola, X. Giannakopoulos, A. Honkela, and J. Karhunen. Nonlinear independent component analysis using ensemble learning: Experiments and discussion. In *Proc. ICA 2000*. In press.

FAST ALGORITHMS FOR BAYESIAN INDEPENDENT COMPONENT ANALYSIS

Harri Valpola and Petteri Pajunen

In *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, (Helsinki, Finland, June 19–22), pp. 233–237, 2000.

Abstract

Fast algorithms for linear blind source separation are developed. The fast convergence is first derived from low-noise approximation of the EM-algorithm given in [1], to which a modification is made that leads as a special case to the FastICA algorithm [2]. The modification is given a general interpretation and is applied to Bayesian blind source separation of noisy signals.

1 Introduction

We consider the problem of finding linearly mixed source signals $\mathbf{s}(t) = [s_1(t), \dots, s_m(t)]^T$ from observed noisy linear mixtures

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t).$$

The mixing matrix $\mathbf{A}$ is unknown and $\mathbf{n}(t)$ is additive noise. When we consider a finite number of observed mixture samples, we may write the data model in matrix form as

$$\mathbf{X} = \mathbf{A}\mathbf{S} + \mathbf{N}$$

Each time-indexed matrix contains a sequence of vector samples, e.g.

$$\mathbf{S} = [\mathbf{s}(1), \mathbf{s}(2), \dots, \mathbf{s}(M)]$$

To find the mixing matrix $\mathbf{A}$, it is necessary that the source signals possess certain statistical properties. For example, it is sufficient that the source signals are not Gaussian. It is also sufficient that the possibly Gaussian sources have time dependencies, together with some other conditions.

2 EM-algorithm for Independent Component Analysis

In the signal model only the vectors $\mathbf{x}(t)$ are observed. Everything else is unknown and must be estimated using the data. In general, the task is to compute the joint posterior distribution for all the unknown parameters conditioned by the mixtures $\mathbf{x}(t)$.

A more simple case is when the maximum likelihood estimate is used for some on the parameters. This can be done by the EM-algorithm where the computation alternates between computing the posterior distribution of one set of variables given the current point estimate of the other set of variables (E-step) and then using the posterior distribution of

the first set of variables to compute a new maximum likelihood estimate of the second set of variables (M-step).

When EM-algorithm is applied to ICA, usually the full posterior distribution is computed for sources and the maximum likelihood estimate is used for the rest of the parameters. This means that in the E-step we need to compute the posterior distribution of the sources $\mathbf{s}$ given $\mathbf{x}$,$\mathbf{A}$ and the noise covariance $\sigma^2\mathbf{I}$

$$p(\mathbf{s}|\mathbf{A}, \mathbf{x}, \sigma^2\mathbf{I})$$

and use it to update our estimates.

Using the matrix notation for the finite number of samples, i.e. $\mathbf{X}$ and $\mathbf{S}$, we can write the M-step (see [3]) re-estimation for the mixing matrix as

$$\hat{\mathbf{A}} = \mathbf{R}_{xs}\mathbf{R}_{ss}^{-1}$$

where the posterior correlation matrices are

$$\mathbf{R}_{xs} = \frac{1}{M}\sum_i \mathbf{x}(i)\,\mathrm{E}\{\mathbf{s}^T(i)|\mathbf{x}(i), \mathbf{A}, \sigma^2\mathbf{I}\} = \mathbf{X}\hat{\mathbf{S}}^T/M$$

$$\mathbf{R}_{ss} = \frac{1}{M}\sum \mathrm{E}\{\mathbf{s}(i)\mathbf{s}^T(i)|\mathbf{x}(i), \mathbf{A}, \sigma^2\mathbf{I}\} = \widehat{\mathbf{S}\mathbf{S}}^T/M\,.$$

The expectations are taken over the posterior distribution of the sources.

We will consider here the case where σ^2 is small. If we further assume that the mixtures are prewhitened, we can constrain the mixing matrix to be orthogonal and we can assume that the sources have unit variance. This makes $\mathbf{R}_{ss}$ a unit matrix.

In [1] the EM-algorithm is derived as a low-noise approximation for the case of square mixing matrix $\mathbf{A}$. First, the posterior mean $p(\mathbf{s}|\mathbf{A}, \mathbf{x}, \sigma^2\mathbf{I})$ is obtained as

$$\hat{\mathbf{s}} = \mathrm{E}\{\mathbf{s}|\mathbf{A}, \mathbf{x}, \sigma^2\mathbf{I}\} \approx \mathbf{s}_0 + \sigma^2(\mathbf{A}^T\mathbf{A})^{-1}f(\mathbf{s}_0)$$

where $f(\cdot)$ is the derivative $\frac{\partial \log p_i(s_i)}{\partial s_i}$ and $\mathbf{s}_0 = \mathbf{A}^{-1}\mathbf{x}$. Since we assumed that the mixing matrix is orthogonal, we can omit the term $(\mathbf{A}^T\mathbf{A})^{-1}$ and we get

$$\hat{\mathbf{s}} = \mathrm{E}\{\mathbf{s}|\mathbf{A}, \mathbf{x}, \sigma^2\mathbf{I}\} \approx \mathbf{s}_0 + \sigma^2 f(\mathbf{s}_0)$$

Substituting the above approximations we get

$$\begin{aligned}\hat{\mathbf{A}} &= \mathbf{X}\hat{\mathbf{S}}^T/M \\ &\approx \mathbf{X}\mathbf{S}_0^T/M + \sigma^2\mathbf{X}\mathbf{F}(\mathbf{S}_0^T)/M \\ &= \mathbf{A} + \sigma^2\mathbf{X}\mathbf{F}(\mathbf{S}_0^T)/M\end{aligned}$$

As the authors mention in [1], this approximation leads to an EM-algorithm which converges slowly with low noise variance σ^2. They also point out that there is no visible "noise-correction". It is precisely this point that we will address in the next section.

3 Fast EM-algorithm by Filtering of Gaussian Noise

With low noise variance σ^2 the convergence of the EM-algorithm to the optimal value takes a time proportional to $1/\sigma^2$. We will next show how the re-estimation step can be modified so that the convergence rate will be independent of σ^2 which yields a significant speedup if σ^2 is small.

Consider the case that we estimate the sources one at a time and that the sources are assumed to be whitened and the mixing matrix $\mathbf{A}$ orthonormal. Denote one of the source signals in the optimal solution as $\hat{\mathbf{s}}_{opt}$. By optimality we mean that the standard EM-algorithm will eventually converge to $\hat{\mathbf{a}}_{opt} = \mathbf{X}\hat{\mathbf{s}}_{opt}^T$ with $\hat{\mathbf{s}}_{opt} = \mathrm{E}\{\mathbf{s}|\hat{\mathbf{a}}_{opt}, \mathbf{X}, \sigma^2\mathbf{I}\}$.

When we have not yet found the optimal vector $\hat{\mathbf{a}}_{opt}$, we have

$$\mathbf{s}_0 = \alpha\mathbf{s}_{opt} + \beta\mathbf{s}_G$$

where $\alpha^2 + \beta^2 = 1$. The noise $\mathbf{s}_G$ is mostly due to the other sources and to a small extent the Gaussian noise in the data. We can think that the E-step filters away the noise by making use of the knowledge about the prior distribution of $\mathbf{s}$. This gives one point of view into the slow convergence: in low noise case most of the unwanted signal $\mathbf{s}_G$ is due to other sources and a slow convergence results. From this point of view, it is obvious that we can speed up the convergence if we can filter away also that part of $\mathbf{s}_G$ which is due to other sources.

When we are far from the optimal solution, it is natural to assume that $\beta \approx 1$ and $\alpha \approx 0$. Since $\mathbf{a}$ and $\mathbf{s}_0$ are linearly related, we get

$$\alpha\hat{\mathbf{a}}_{opt} = \hat{\mathbf{a}} - \beta\hat{\mathbf{a}}_G \approx \hat{\mathbf{a}} - \hat{\mathbf{a}}_G.$$

If we can compute $\mathbf{a}_G$, we can adjust the vector $\mathbf{a}$ to take into account the apparent noise due to other sources. By the central limit theorem, the distribution of the sum of contributions from several other sources approaches Gaussian as the number of other sources increases. This leads to the following modification: we may estimate $\hat{\mathbf{a}}_G$ using the same re-estimation whose result will be approximately

$$\hat{\mathbf{a}}_G \approx \mathbf{a} + \sigma^2\mathbf{X}_G\mathbf{F}(\mathbf{s}_{0G})/M.$$

where $\mathbf{X}_G$ is the set of mixtures replaced by Gaussian noise with the same covariance as $\mathbf{X}$ and $\mathbf{s}_{0G}$ is the source obtained as $\mathbf{a}^T\mathbf{X}_G$. The Gaussian source $\mathbf{s}_{0G}$ is the projection of Gaussian noise to the subspace spanned by $\mathbf{a}$ and therefore represents the contribution of the other sources and some Gaussian noise to the estimated source $\mathbf{s}_0$. As derived above, we can eliminate much of this noise by updating $\mathbf{a}$ using the difference $\hat{\mathbf{a}} - \hat{\mathbf{a}}_G$ which is then normalized. The normalization can be done, since scaling of the sources is an undeterminacy in ICA.

Taking the difference yields approximately

$$\hat{\mathbf{a}} - \hat{\mathbf{a}}_G \approx \sigma^2[\mathbf{X}\mathbf{F}(\mathbf{s}_0)^T - \mathbf{X}_G\mathbf{F}(\mathbf{s}_{0G})^T]/M$$

which shows that the normalization cancels the effect of σ^2 from the learning rule:

$$\hat{\mathbf{a}}_{new} = \frac{\hat{\mathbf{a}} - \hat{\mathbf{a}}_G}{\|\hat{\mathbf{a}} - \hat{\mathbf{a}}_G\|}\,.$$

We assumed above that there was a lot of Gaussian noise by approximating $\beta \approx 1$. It turns out that the above modification does not affect the optimal solutions of the algorithm, i.e., if $\hat{\mathbf{a}}_{opt}$ is a fixed point of the original EM-algorithm, it is also a fixed point of the modified algorithm. This follows immediately from the fact that $\hat{\mathbf{a}}_G$ is always parallel to

$\mathbf{a}$ since $\mathbf{X}_G$ is spherically symmetric. To get a rough idea about why this is so, suppose there is a vector $\mathbf{b}$ which is orthogonal to $\mathbf{a}$, i.e., $\mathbf{b}^T\mathbf{a} = 0$. Then

$$\mathbf{b}^T\hat{\mathbf{a}}_G \approx \mathbf{b}^T\mathbf{a} + \mathbf{b}^T\mathbf{X}_G\mathbf{F}(\mathbf{s}_{0G}) = \mathbf{s}'_{0G}\mathbf{F}(\mathbf{s}_{0G}) = 0\,.$$

The last step follows from the fact that $\mathbf{s}'_{0G}$ is a projection to an orthogonal direction form $\mathbf{s}_{0G}$ and by Gaussianity of $\mathbf{X}_G$, statistically independent form $\mathbf{F}(s_{0G})$. But since this must hold for all $\mathbf{b}$ which are orthogonal to $\mathbf{a}$, it follows that $\hat{\mathbf{a}}_G$ has to be parallel to $\mathbf{a}$.

In the next section we add validity to the result by showing that FastICA algorithm follows from this procedure.

4 FastICA as EM-Algorithm with Filtering of Gaussian Noise

The FastICA algorithm [2] can be interpreted as performing the above described noise removal. In FastICA the requirement of whitening the sources is also made and therefore $\mathbf{R}_{ss} = \mathbf{I}$ and $(\mathbf{A}^T\mathbf{A})^{-1} = \mathbf{I}$. Then, the sources can be found one by one and we can consider a single column $\mathbf{a}$ of the mixing matrix $\mathbf{A}$.

To derive the FastICA algorithm from the modified EM-algorithm, it is sufficient to note that the term $\mathbf{X}_G\mathbf{F}(\mathbf{s}_{0G})^T/M = \mathbf{a}\mathbf{s}_{0G}F(\mathbf{s}_{0G})^T/M$ is $C_f\mathbf{a}$ where C_f is a constant that depends only on the nonlinear function $f(\cdot)$. Then the update rule is

$$\hat{\mathbf{a}} - \hat{\mathbf{a}}_G = \mathbf{X}\mathbf{F}(\mathbf{s}_0^T) - C_f\mathbf{a}$$
$$\hat{\mathbf{a}}_{new} = \frac{\hat{\mathbf{a}} - \hat{\mathbf{a}}_G}{\|\hat{\mathbf{a}} - \hat{\mathbf{a}}_G\|}$$

which is the FastICA algorithm, where the constant C_f is the expectation $\mathrm{E}\{s_{0G}f(s_{0G})\}$.

The choice of fixed nonlinearity $f(\cdot)$ is implicitly connected to the distribution of the sources s. The derivation of the EM-algorithm required that

$$f(s) = \frac{\partial \log p(s)}{\partial s}$$

However, we see that $f(\cdot)$ has certain degrees of freedom due to taking the difference $\mathbf{X}\mathbf{F}(\mathbf{s}_0^T) - \mathbf{X}_G\mathbf{F}(\mathbf{s}_{0G}^T)$. Expanding f polynomially we obtain $p(s) = \exp(a + bs + cs^2 + dg(s))$ where $g'(s) = f(s)$ and $g(s)$ contains all the powers of f higher than two and possibly lower moments too. This representation follows since in the update rule constants and linear terms of $f(\cdot)$ will cancel out. Therefore they will appear in the distribution $p(s)$ in the exponent with the power raised by one due to integration. Since $p(s)$ must be a probability density, the constant a will be fixed by the requirement $\int p(s)ds = 1$. Mean and variance of s will determine the constants b and c, since the sources are required to be zero-mean and whitened (variance is fixed to unity). There is one free parameter d left, which means that there is not only one distribution corresponding to $f(\cdot)$ but a family of $p(s)$. Typically the family includes both super- and sub-Gaussian densities, which is why the same $f(\cdot)$ can be used for both cases.

5 Application to General ICA Algorithms

The procedure giving faster convergence derived in previous sections is a general approach and FastICA was seen to be a special case. Since the faster convergence was achieved by comparing the re-estimation step to Gaussian noise removal, the approach is valid for any situation where the general noisy ICA model holds with Gaussian noise and linear mixtures. It is not required that the E-step uses the approximation $\hat{\mathbf{s}} \approx \mathbf{s}_0 + \sigma^2 f(\mathbf{s}_0)$; instead, it can be any method that can use $\mathbf{s}_0$ to compute $\hat{\mathbf{s}}$. Denote this estimation by

$$\hat{\mathbf{s}} = \mathbf{g}(\mathbf{s}_0) \,.$$

Then it is always possible to replace the source with Gaussianized source $\mathbf{s}_{0G}$ and obtain

$$\hat{\mathbf{s}}_G = \mathbf{g}(\mathbf{s}_{0G}) \,.$$

Having estimated two sets of sources, we can apply any method whatsoever to estimate the mixing matrix using the newly estimated sources. This gives us two new estimates of the vectors $\hat{\mathbf{a}}$ and $\hat{\mathbf{a}}_G$ of the mixing matrix. The final estimate is obtained as the normalized difference as above.

6 Application to Bayesian Noisy ICA

Above, noise was assumed to have a small variance to justify certain approximations. Therefore the result was not strictly an algorithm for noisy ICA since the approximations get worse with increasing noise variance. Below, we will consider a speedup modification for Bayesian noisy ICA. The Bayesian approach adopted here gives certain important advantages:

- noise can have a finite variance
- source densities need not be fixed a priori; they can be estimated
- the number of sources can be estimated
- model comparison is possible

Specifically the source distributions are modeled as mixtures of Gaussians and the posterior is approximated using ensemble learning. The treatment of the source distribution is similar to [4] which uses a factorial approximation of the posterior source distributions in connection with the EM-algorithm. The modification would be directly applicable to the algorithm in [4], but we will consider an algorithm where all the posterior distribution is estimated for all parameters, i.e., point estimates are not used at all.

6.1 Overview of the Bayesian ICA Algorithm

In [5], it is described how to use ensemble learning for the noisy ICA model. The posterior distribution is over all unknown parameters, including the mixing matrix $\mathbf{A}$. In ensemble learning, a factorial approximation $q(\mathbf{S}, \mathbf{A}, \dots)$ is fitted to the actual posterior distribution $p(\mathbf{S}, \mathbf{A}, \dots | \mathbf{X})$ by minimising the Kullback-Leibler information between them, i.e., the cost function which is minimized during learning is

$$I(q;p) = \mathrm{E}_q\{\log(q/p)\} \,.$$

The algorithm is computationally efficient when the approximation $q(\cdot)$ of the posterior probability $p(\cdot|\mathbf{X})$ is chosen to be factorial. This can be seen as an extension of the factorial

EM-algorithm in [4], where $q(\cdot)$ included only the posterior distribution of the sources. For further details, see for instance [6, 7], where ensemble learning is applied to nonlinear ICA.

The ICA algorithm based on ensemble learning works in much the same way as EM-algorithm. First the distribution of the sources is computed by using the current estimate of the distribution of the other parameters. Then the distribution of the other parameters is computed using this distribution of the sources. The posterior distributions of the parameters are approximated by Gaussian distribution which means that for each element of the mixing matrix $\mathbf{A}$, the posterior mean and variance is estimated. The modification will be applied to the posterior mean of the mixing matrix.

For each vector of the mixing matrix, the modified posterior mean will be the normalized difference between the posterior mean estimated from the original sources and the Gaussianized sources. The iteration is then repeated by estimating the posteriors of the sources again, using the new parameter distribution.

In practice, the algorithm is performed in deflatory manner, that is, the sources are extracted one by one. The mixtures are prewhitened and then the mixing matrix is estimated one column $\mathbf{a}$ at a time.

A heuristic stabilization is added to ensure convergence. This is achieved by updating the vector $\mathbf{a}$ to be a linear combination $\alpha\mathbf{a}_{new} + (1-\alpha)\mathbf{a}_{old}$. The coefficient α is increased when consecutive corrections to $\mathbf{a}$ have a positive inner product which means that they do not change to opposite directions. Otherwise, α is decreased.

7 Experiments

The Bayesian ICA algorithm was tested on MEG data, which is identical to the data used in [8]. The data has 122 channels of measurements over two minutes digitized at 148 Hz. The measurements contain signals resulting in the electrical activity of the brain but also signals which can be considered artifacts. These include signals caused by muscular activity, eye movements, cardiac rhythm and even a signal caused by a digital watch that the test subject was wearing. Since it can be assumed that most of the artifacts are independent of the brain activity, it is hoped that ICA can find the artifacts.

The Bayesian ICA algorithm was used to separate 30 sources from the 122 measures channels. The results obtained were comparable to those reported in [8]. In figure 1, five measurements and five non-Gaussian sources found by the algorithm are illustrated.

The modification of the estimate of $\mathbf{a}$ by the estimate $\mathbf{a}_G$ typically reduces the convergence time by a factor of ten; the iteration typically converged in 30 iterations.

8 Discussion

First we considered the EM-algorithm for finding independent components with low noise. The problem of slow convergence was noted and an improvement was proposed. When finding the sources one at a time, the contributions of the unwanted sources was treated as noise, which leads to faster convergence. Although the approach was found to be implicitly the same as in the FastICA algorithm, it is valid for other situations too. In Bayesian ICA for i.i.d. sources, the modification can be applied as proposed. Other possibilities include finding groups of components that are not mutually independent but are independent related to other components not in the group. The independent components are then projections to multidimensional subspaces instead of one-dimensional projections. This has

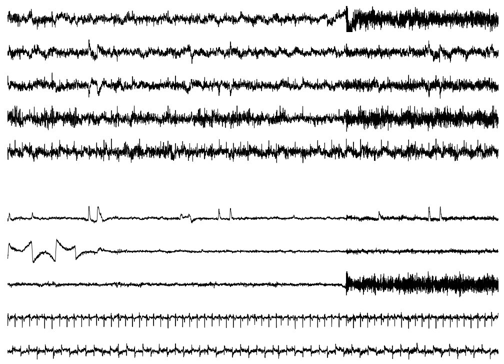

Figure 1: Above: five MEG measurements. Below: five separated sources found by the Bayesian ICA algorithm.

been proposed e.g. in [9, 10]. The modification proposed in this paper applies to this case too, since the contributions of sources not in the group can be regarded as approximately Gaussian noise.

Further work includes finding more general principles, where the modification could be derived for other cases, such as time-dependent sources or nonlinear ICA.

References

[1] O. Bermond and J. Cardoso, "Approximate likelihood for noisy mixtures," in *Proc. Int. Workshop on Independent Component Analysis and Signal Separation (ICA'99)*, (Aussois, France), pp. 325–330, January 1999.

[2] A. Hyvärinen and E. Oja, "A fast fixed-point algorithm for independent component analysis," *Neural Computation*, vol. 9, no. 7, pp. 1483–1492, 1997.

[3] E. Moulines, J. Cardoso, and E. Gassiat, "Maximum likelihood for blind separation and deconvolution of noisy signals using mixture models," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 3617–3620, 1997.

[4] H. Attias, "Independent factor analysis," *Neural Computation*, vol. 11, no. 4, pp. 803–851, 1999.

[5] H. Lappalainen, "Ensemble learning for independent component analysis," in *Proc. Int. Workshop on Independent Component Analysis and Signal Separation (ICA'99)*, (Aussois, France), pp. 7–12, January 1999.

[6] H. Lappalainen, "Nonlinear independent component analysis using ensemble learning: Theory," in *Proc. Int. Workshop on Independent Component Analysis and Signal Separation (ICA2000)*, (Helsinki, Finland), June 2000.

[7] H. Lappalainen, X. Giannakopoulos, A. Honkela, and J. Karhunen, "Nonlinear independent component analysis using ensemble learning: Experiments and discussion," in *Proc. Int. Workshop on Independent Component Analysis and Signal Separation (ICA2000)*, (Helsinki, Finland), June 2000.

[8] R. Vigario, V. Jousmäki, M. Hämäläinen, R. Hari, and E. Oja, "Independent component analysis for identification of artifacts in magnetoencephalographic recordings," in *Advances in Neural Information Processing Systems 10 (NIPS'97)*, (Cambridge, MA), pp. 229–235, MIT Press.

[9] J.-F. Cardoso, "Multidimensional independent component analysis," in *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP'98)*, vol. 4, (Seattle, Washington, USA), pp. 1941–1944, May 1998.

[10] A. Hyvärinen and P. O. Hoyer, "Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces," *Neural Computation*, 2000. (in press).

UNSUPERVISED LEARNING OF NONLINEAR DYNAMIC STATE-SPACE MODELS

Harri Valpola

Publications in Computer and Information Science A59, Helsinki University of Technology, Espoo, Finland, 2000.

Abstract

This technical report describes how the nonlinear factor analysis algorithm introduced in [4] can be extended by taking into account the dynamics of the factors. The resulting algorithm represents the observations as having been generated by a nonlinear dynamical system. The internal states and the nonlinear mappings describing the nonlinear system are assumed to be unknown and they are estimated from the observations. Experiments with artificial data verify that the algorithm is able to reconstruct the dynamics of the underlying process which has generated the observations.

1 Introduction

This technical report describes how the nonlinear factor analysis (NLFA) algorithm [4] can be extended by modelling the dynamics of the factors. The goal is to find factors which not only represent the observations compactly but are also predictable. Only those parts which differ from the NLFA algorithm are explained and this report should therefore be read together with [4].

Like in NLFA, learning is unsupervised and is based on uncovering regularities in the observations. NLFA can capture static regularities within observation vectors $\mathbf{x}(t)$ but discards any temporal structure present in the sequence of observations. Including a model of the dynamics of the factors results in a nonlinear dynamic factor analysis (NDFA) model which can capture both static and temporal structure of the observations.

The generative model for the observations $\mathbf{x}(t)$ is as follows:

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t) \tag{1}$$

$$\mathbf{s}(t) = \mathbf{g}(\mathbf{s}(t-1)) + \mathbf{m}(t) \tag{2}$$

The observations $\mathbf{x}(t)$ are assumed to have been generated by the factors $\mathbf{s}(t)$ through a nonlinear mapping $\mathbf{f}$. It is reasonable to assume that the model misses some of the factors affecting the observations and that the nonlinear mapping is somewhat inaccurate. The error caused by these imperfections is modelled by i.i.d. Gaussian noise $\mathbf{n}(t)$. The nonlinear mapping $\mathbf{f}$ is modelled by a multi-layer perceptron (MLP) network.

The model for the dynamics of the factors has almost the same structure as the observation model. The observations are assumed to have been generated by the factors at the previous time instant. This means that the factors can be interpreted as the states of a dynamical system. The noise $\mathbf{m}(t)$ of the dynamic model is often called process noise or innovation process. Similar models have been proposed in [2, 7] where the nonlinear mappings are modelled by radial basis functions [6] and in [1] where the nonlinearities are modelled by MLP networks.

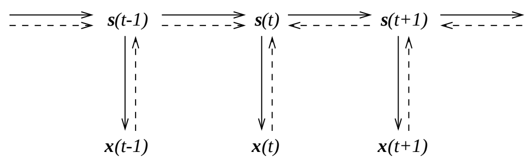

Figure 1: The causal relations assumed by the model are depicted by solid arrows. Observations $\mathbf{x}(1)$, ..., $\mathbf{x}(t)$, ... give information about the values of the factors. The flow of information to the factor $\mathbf{s}(t)$ is represented by dashed arrows. Bayes' rule is used for reversing the arrows.

As the dynamic model (2) has the same functional form as observation model (1), similar sets of hyperparameters can be used for both models. Learning could also be achieved with the same algorithm which was used for NLFA in [4]. Some minor changes are made, however, which take into account the fact that the dynamic mapping $\mathbf{g}$ can usually be expected to be closer to identity mapping than zero and the model of the dynamics induces posterior correlations to the factors.

Section 2 discusses the properties of the state-space model used in this report and some of its alternatives. Section 3 introduces the modifications to the learning algorithm. Results of simulations are reported in section 4.

2 Predictable factors and state-space models

Models are tools which enable making inferences based on observations. One of their most important applications is prediction. The model can be used to infer the expected state of the world in the future or predict the expected consequences of various actions.

In unsupervised learning, the goal is to find a compact representation for the observations. The benefit is that it is often easier to find the connection between two subsets of observations using the compact representations than directly between the observations. In this case we are interested in the connection which can be made in the temporal domain.

NLFA can find compact representation for the observations. It does not take into account the temporal behaviour of the observations, but it can be expected that it is often easier to predict the future factors from the past factors than directly the future observations from the past observations. It would therefore be possible to first use NLFA to find a compact representation for observations $\mathbf{x}(t)$ in terms of factors $\mathbf{s}(t)$ and then find the mapping from the past factors $\mathbf{s}(t-1)$, $\mathbf{s}(t-2)$, ..., to the current factor $\mathbf{s}(t)$.

The drawback of this approach would be that in the first stage, learning does not explicitly aim at finding factors which facilitate the prediction. This can be remedied simply by combining the stages and letting the learning of factors take into account both the accuracy of description of the observations and the prediction of future factors.

The model defined by (1) and (2) does exactly this. Learning of factors $\mathbf{s}(t)$ takes into account three sources of information: 1) the factors should be able to represent the observations $\mathbf{x}(t)$, 2) the factors should be able to predict the factors $\mathbf{s}(t+1)$ at the next

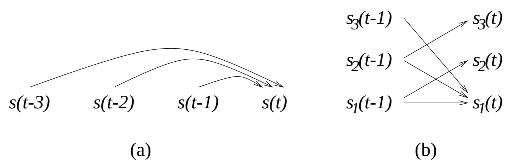

Figure 2: (a) The factor $s(t)$ depends on the three previous values $s(t-3)$, $s(t-2)$ and $s(t-1)$. (b) This dynamics can be transformed into an equivalent state representation where $s_1(t)$ corresponds to $s(t)$ while $s_2(t)$ and $s_3(t)$ store the values $s(t-1)$ and $s(t-2)$, respectively.

time step and 3) the factors should be well predicted by the factors $\mathbf{s}(t-1)$ at the previous time step. This is depicted in figure 1.

As (2) shows, the model assumes that the factors $\mathbf{s}(t)$ can be predicted from the immediately preceding factors $\mathbf{s}(t-1)$ without knowing the factors in the more distant past. This does not restrict the class of dynamical processes which can be modelled because any model with long range dependencies can be converted into an equivalent model with only one step dependencies but larger number of factors. This means that the factors store all the information needed for predicting the dynamic behaviour of the process and therefore the factors can be interpreted as the state of the dynamical system.

Figure 2 gives an example about how a model with three step delays can be transformed into an equivalent state representation with only one step delays but more factors. In this case the two extra factors $s_2(t)$ and $s_3(t)$ store the values $s(t-1)$ and $s(t-2)$, but they could, for instance, store the first and second time derivatives of $s(t)$ as well. The benefit of using the state representation is that the model can learn the structure of the memory.

3 Model and learning algorithm

The functional form and learning algorithm for the nonlinear dynamic factor analysis (NDFA) model proposed in this report are very close to the ones introduced for NLFA in [4]: the mappings $\mathbf{f}$ and $\mathbf{g}$ are modelled by MLP networks and a gradient based learning algorithm is used for iteratively updating the approximation of the posterior probability of the unknown variables. There are some modifications, however, which take into account the fact that the mapping $\mathbf{g}$ models the dynamics of the factors.

First of all, it can be expected that the values of the factors $\mathbf{s}(t)$ are close to the values at the previous time step $\mathbf{s}(t-1)$. This prior knowledge can be taken into account by using the MLP network to model only the change in factors instead of trying to learn the whole mapping from $\mathbf{s}(t-1)$ to $\mathbf{s}(t)$. The functional form of the dynamic mapping $\mathbf{g}$ is thus

$$\mathbf{g}(s(t-1)) = \mathbf{s}(t-1) + \mathbf{D}\tanh[\mathbf{C}\mathbf{s}(t-1) + \mathbf{c}] + \mathbf{d}\,, \tag{3}$$

while the observation mapping is

$$\mathbf{f}(s(t)) = \mathbf{B}\tanh[\mathbf{A}\mathbf{s}(t) + \mathbf{a}] + \mathbf{b}\,. \tag{4}$$

It should be noted that the MLP network modelling the dynamics can learn to overrule the term $\mathbf{s}(t-1)$ if it turns out that the value of the factor at time $t-1$ does not bear information about the value of the factor at time t.

3.1 Improved approximation of the posterior probability

In [4], the posterior probability of the unknown variables was approximated as Gaussian distribution with diagonal covariance matrix. This means that the unknown variables were approximated to be independent given the observations. Notice that this assumption is false even if the unknown variables are assumed to be independent a priori. For instance, the factors are assumed to be independent a priori but observations induce dependencies between them. These dependencies are strongest for factors at the same time instant, that is, $s_i(t)$ and $s_j(t)$ have a posterior dependence but $s_i(t_1)$ and $s_j(t_2)$ can be nearly independent when t_1 is far from t_2.

The NLFA model has an indeterminacy of the rotation of the factors and the model can utilise this by choosing the rotation which makes the factors independent not only a priori but also a posteriori. However, the inclusion of the dynamic model causes posterior dependencies between factors at different time steps and these dependencies do not vanish for any rotation or any other mapping of the factor space. The smaller the process noise the stronger the dependence will be.

Taking into account the full posterior covariance between $\mathbf{s}(t-1)$ and $\mathbf{s}(t)$ is computationally costly if the dimension of the factor space is large. In practice, the most significant posterior correlations are the posterior autocorrelations of the factors, that is, the correlations between $s_i(t-1)$ and $s_i(t)$. They can be taken into account without increasing the computational complexity significantly.

In [4], the approximation of the posterior probability of the factors had the factorial form

$$q(\mathbf{S}|\mathbf{X}) = \prod_{i,t} q(s_i(t)|\mathbf{X})\,, \tag{5}$$

where $\mathbf{S}$ denotes the factors and $\mathbf{X}$ the observations. The approximations $q(s_i(t)|\mathbf{X})$ were Gaussian with mean $\bar{s}_i(t)$ and variance $\tilde{s}_i(t)$. For the dynamical model introduced in this report, the approximation has the form

$$q(\mathbf{S}|\mathbf{X}) = \prod_{i,t} q(s_i(t)|s_i(t-1), \mathbf{X})\,, \tag{6}$$

where $q(s_i(t)|s_i(t-1), \mathbf{X})$ is Gaussian and depends linearly on $s_i(t-1)$:

$$q(s_i(t)|s_i(t-1), \mathbf{X}) = N(s_i(t)|\bar{s}_i(t) + \breve{s}_i(t,t-1)[s_i(t-1) - \bar{s}_i(t-1)], \mathring{s}_i(t))\,. \tag{7}$$

Here $N(\xi|\mu, \sigma^2)$ denotes a Gaussian distribution over ξ with mean μ and variance σ^2.

Given $s_i(t-1)$, the posterior variance of $s_i(t)$ is $\mathring{s}_i(t)$ and the posterior mean is $\bar{s}_i(t) + \breve{s}_i(t,t-1)[s_i(t-1) - \bar{s}_i(t-1)]$. The approximate posterior $q(s_i(t)|s_i(t-1), \mathbf{X})$ is thus parametrised by the mean $\bar{s}_i(t)$, linear dependence $\breve{s}_i(t,t-1)$ and variance $\mathring{s}_i(t)$ as defined by (7), whereas in NLFA the posterior of the factor was parametrised by mean and variance alone.

It is easy to see by induction that if the past values of the factors are marginalised out, the posterior mean of $s_i(t)$ is $\bar{s}_i(t)$ and posterior variance $\tilde{s}_i(t)$ is

$$\tilde{s}_i(t) = \mathring{s}_i(t) + \breve{s}_i^2(t,t-1)\tilde{s}_i(t-1)\,. \tag{8}$$

Notice that the marginalised variances $\tilde{s}_i(t)$ are computed recursively in a sweep forward in time.

3.2 Feedforward and backward computations

The feedforward computations start with the parameters of the posterior approximation of the unknown variables of the model. For the factors, the parameters of the posterior approximation are the posterior mean $\bar{s}_i(t)$, the posterior variance $\mathring{s}_i\ (t)$ and the dependence $\breve{s}_i(t, t-1)$. The end result of the feedforward computations is the value of the cost function C.

The first stage of the computations is the iteration of (8) to obtain the marginalised posterior mean $\bar{s}_i(t)$ and variance $\tilde{s}_i(t)$ of the factors. Thereafter the computations proceed like in the NLFA algorithm: the means and variances are propagated through the MLP networks. The final stage, the computation of the cost function, differs only in the terms $\int q(s_i(t)|s_i(t-1), \mathbf{X}) \ln q(s_i(t)|s_i(t-1), \mathbf{X}) ds_i(t)$ and $-\int q(\boldsymbol{\theta}|\mathbf{X}) \ln p(s_i(t)|\boldsymbol{\theta}) d\boldsymbol{\theta}$. In the NLFA algorithm, the former had the form

$$\ln 2\pi e \tilde{s}_i(t)\,, \tag{9}$$

but now they have the form

$$\ln 2\pi e \ \mathring{s}_i\ (t)\,. \tag{10}$$

The latter terms can be shown to yield

$$\frac{1}{2}\left[(\bar{s}_i(t) - \bar{g}_i(t))^2 + \mathring{s}_i\ (t) + \left(\breve{s}_i(t, t-1) - \frac{\partial g_i(t)}{\partial s_i(t-1)}\right)^2 \tilde{s}_i(t-1) + \tilde{g}_i^*(t)\right] e^{2\tilde{v}_i - 2\bar{v}_i} + \\ \bar{v}_i + \frac{1}{2}\ln 2\pi\,, \tag{11}$$

where the ith component of the vector $\mathbf{g}(\mathbf{s}(t-1))$ is denoted by $g_i(t)$ and the variance parameter of the ith factor by v_i, and by $\tilde{g}_i^*(t)$, the posterior variance of $g_i(t)$ without the contribution of $s_i(t-1)$, that is, assuming $s_i(t-1)$ fixed. Notice that if $\breve{s}_i(t, t-1)$ is zero, the term inside the square brackets takes the form $(\bar{s}_i(t) - \bar{g}_i(t))^2 + \mathring{s}_i\ (t) + \tilde{g}_i(t)$ because $\tilde{g}_i^*(t)$ is defined to be $\tilde{g}_i(t) - [\partial g_i(t)/\partial s_i(t-1)]^2 \tilde{s}_i(t-1)$.

In the feedbackward phase, the gradient of the cost function C w.r.t. the parameters of the posterior approximation is computed by the back-propagation algorithm, that is, the steps of the feedforward computations are reversed and the gradient of the cost function is propagated backwards to the parameters of the posterior approximation. Since the essential modification to the feedforward phase of NLFA algorithm is (8), this is also the essential modification in the backward computations.

The cost function is a function of parameters of the posterior approximation. In the computation of the cost function, the marginalised posterior variances $\tilde{s}_i(t)$ of the factors are used as intermediate variables and hence the gradient is also computed through these variables. Let us use the notation $C(\tilde{s}_i(t))$ to mean that C is considered to be a function of the intermediate variables $\tilde{s}_i(1), \ldots, \tilde{s}_i(t)$ in addition to the parameters of the posterior approximation. The gradient computations resulting from (8) by the chain rule are then as follows:

$$\frac{\partial C}{\partial \ \mathring{s}_i\ (t)} = \frac{\partial C(\tilde{s}_i(t))}{\partial \ \mathring{s}_i\ (t)} + \frac{\partial C(\tilde{s}_i(t))}{\partial \tilde{s}_i(t)} \frac{\partial \tilde{s}_i(t)}{\partial \ \mathring{s}_i\ (t)} = \frac{\partial C(\tilde{s}_i(t))}{\partial \ \mathring{s}_i\ (t)} + \frac{\partial C(\tilde{s}_i(t))}{\partial \tilde{s}_i(t)} \tag{12}$$

$$\frac{\partial C}{\partial \breve{s}_i(t, t-1)} = \frac{\partial C(\tilde{s}_i(t))}{\partial \breve{s}_i(t, t-1)} + \frac{\partial C(\tilde{s}_i(t))}{\partial \tilde{s}_i(t)} \frac{\partial \tilde{s}_i(t)}{\partial \breve{s}_i(t, t-1)} = \\ \frac{\partial C(\tilde{s}_i(t))}{\partial \breve{s}_i(t, t-1)} + 2\frac{\partial C(\tilde{s}_i(t))}{\partial \tilde{s}_i(t)} \breve{s}_i(t, t-1)\tilde{s}_i(t-1) \tag{13}$$

$$\frac{\partial C(\tilde{s}_i(t))}{\partial \tilde{s}_i(t)} = \frac{\partial C(\tilde{s}_i(t+1))}{\partial \tilde{s}_i(t)} + \frac{\partial C(\tilde{s}_i(t+1))}{\partial \tilde{s}_i(t+1)} \frac{\partial \tilde{s}_i(t+1)}{\partial \tilde{s}_i(t)} = \\ \frac{\partial C(\tilde{s}_i(t+1))}{\partial \tilde{s}_i(t)} + \frac{\partial C(\tilde{s}_i(t+1))}{\partial \tilde{s}_i(t+1)} \breve{s}_i^2(t+1, t) \tag{14}$$

The terms $\partial C(\tilde{s}_i(t))/\partial \ \mathring{s}_i\ (t)$ and $\partial C(\tilde{s}_i(t))/\partial \breve{s}_i(t, t-1)$ can be computed from (10) and (11) while $\partial C(\tilde{s}_i(t+1))/\partial \tilde{s}_i(t)$ also includes terms originating from the mappings $\mathbf{f}$ and $\mathbf{g}$ as their feedforward computation starts with the posterior means $\bar{s}_i(t)$ and variances $\tilde{s}_i(t)$.

In the adaptation, the posterior means $\bar{s}_i(t)$ of the factors are treated as in the NLFA algorithm except for the correction in the step size which is discussed in section 3.3. The variances $\mathring{s}_i\ (t)$ are adapted like $\tilde{s}_i(t)$ in the NLFA. The posterior dependence $\breve{s}_i(t, t-1)$ is adapted by solving $\partial C/\partial \breve{s}_i(t, t-1) = 0$ which yields

$$\breve{s}_i(t, t-1) = \frac{\frac{\partial g_i(t)}{\partial s_i(t-1)} e^{2\tilde{v}_i - 2\bar{v}_i}}{2\frac{\partial C(\tilde{s}_i(t))}{\partial \tilde{s}_i(t)} + e^{2\tilde{v}_i - 2\bar{v}_i}}\,. \tag{15}$$

Equation (15) shows that $\breve{s}_i(t, t-1)$ depends on $\partial C(\tilde{s}_i(t))/\partial \tilde{s}_i(t)$ which in turn depends on $\breve{s}_i(t+1, t)$ as (14) shows. This means that the update of the dependencies $\breve{s}_i(t, t-1)$ and the computation of the gradient w.r.t. the marginalised variance $\tilde{s}_i(t)$ are done recursively backward in time which is the counterpart of (8) where the marginalised variances are computed recursively forward in time.

3.3 Correction of the step size

In the NLFA algorithm, the step sizes of the updates of the posterior means $\bar{s}_i(t)$ of the factors were corrected by taking into account the correlated effects of the updates. In NLFA the update of the posterior means $\bar{s}_i(t)$ affect the cost function through terms which result from the prior for factor $s_i(t)$ and likelihood $p(\mathbf{x}(t)|\mathbf{s}(t))$. The likelihood results in several terms, one for each component of the observation vector $\mathbf{x}(t)$. The problem was that the update rule gives the optimal value for each $\bar{s}_i(t)$ assuming that otherwise the posterior approximation stays fixed but this assumption is violated because for efficiency, all posterior means and variances are updated at once. This was remedied by utilising the Jacobian matrix of $\mathbf{f}(\mathbf{s}(t))$ w.r.t. $\mathbf{s}(t)$.

In the NDFA algorithm the situation is slightly more complex as the "prior" $p(\mathbf{s}(t)|\mathbf{s}(t-1))$ of the factors $\mathbf{s}(t)$ is also affected by the factors at the previous time step, but essentially the same procedure can be used: first the effect of all the proposed updates of the posterior means of factors on $\mathbf{x}(t)$ and $\mathbf{s}(t)$ is computed, then the step is projected back to the factors. This is compared with the result which would be obtained assuming that each of the posterior means of the factors is the only parameter of the posterior approximation to change. The size of the update is then corrected by the ratio of the assumed and actual effect of the proposed updates just as in the NLFA algorithm.

3.4 Initialisation of the factors

In the NLFA algorithm, the factors were initialised to the principal components of the observations. This could also be used in NDFA, but the drawback would be that the initialisation would only aim at factors which give a good representation for the observations but would not explicitly aim at factors which can predict the future factors and can themselves be predicted from the past factors.

Phase space embedding methods are standard techniques in the analysis of nonlinear dynamical systems and they can also be applied here. In short, the idea is that the internal state of a (deterministic) dynamical system is embedded in the sequence of observations. It may be impossible to deduce the state $\mathbf{s}(t)$ of the system from one measurement $\mathbf{x}(t)$ alone. Under suitable conditions, however, a sequence $[\mathbf{x}(t)\, \mathbf{x}(t-1)\, \mathbf{x}(t-2) \ldots \mathbf{x}(t-D)]$ of observations contains all the information needed to reconstruct the original state if the number D of delays is large enough [8].

The solution used here is to initialise the factors to principal components of a sequence of observations. In general, the state of a dynamical system is nonlinearly embedded in the sequence, but principal component analysis can nevertheless find a good starting point for the factors. To be more specific, instead of computing the principal components from $\mathbf{x}(t)$, they are computed from $\mathbf{y}(t) = [\mathbf{x}^T(t)\; \mathbf{x}^T(t-1)\; \ldots\; \mathbf{x}^T(t-D)]^T$.

The NLFA algorithm could be used[1] for extracting a state which is nonlinearly embedded in $\mathbf{y}(t)$. This would amount to computing the principal components of $\mathbf{y}(t)$, then using the NLFA algorithm to further refine the extracted factors and finally learning the dynamic model for $\mathbf{x}(t)$ starting from the factors given by the NLFA algorithm. However, here the dynamic model is included already in the second phase: the NDFA algorithm is used for finding factors which can represent the concatenated observations $\mathbf{y}(t)$.

The benefit of this procedure is that using the concatenated observations $\mathbf{y}(t)$ as observations promotes the algorithm to find factors which can represent not only the original observations $\mathbf{x}(t)$ but also their time behaviour. Once these factors have appeared and the dynamic mapping $\mathbf{g}(t)$ has adapted, the factors representing the dynamics have support from the dynamic mapping and the time lagged part of $\mathbf{y}(t)$ can be dropped away leaving only $\mathbf{x}(t)$. If the learning starts directly with $\mathbf{x}(t)$, there is the danger that some of the factors describing the dynamics will be effectively pruned away.

4 Results

4.1 Problem setting

The working hypothesis to be tested was that the NDFA algorithm should be able to find a representation for the observations which facilitates the prediction of the observations, i.e., a representation where modelling the dynamics of the underlying data generating process is easier than directly for the original observations.

Data was generated by first defining the underlying dynamic process and then mapping some of the states of the process onto observations by one of the random MLP networks which were used in [4].

The underlying dynamic process was constructed by combining three independent dynamic processes. One of the processes was harmonic oscillator with angular velocity 1/3. The harmonic oscillator has a two-dimensional state representation and linear dynamics.

[1]This was suggested by Dr. J. P. Barnard (personal communication).

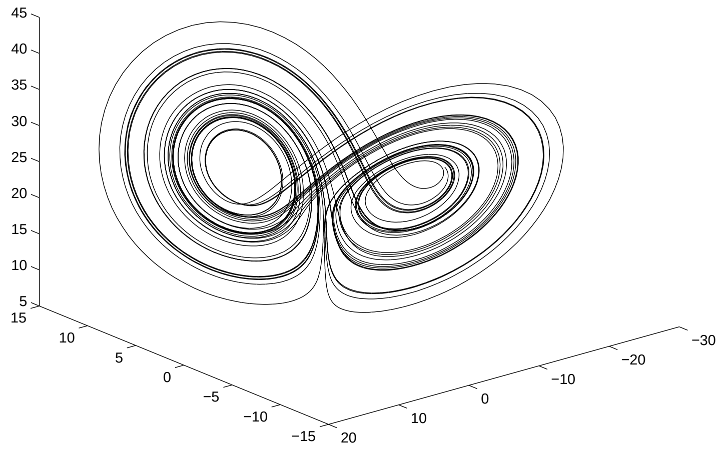

Figure 3: Time evolution of the state resulting from the Lorenz system with $\sigma = 3$, $\rho = 26.5$ and $\beta = 1$.

Notice, however, that if the harmonic oscillator is modelled with linear dynamics, the gain has to be exactly one. Otherwise the oscillations will either decay or grow exponentially for gains less than or greater than one, respectively. A robust model for a harmonic oscillator with a constant amplitude therefore needs to be nonlinear.

The two other processes were chosen to be Lorenz processes [5]. The Lorenz system has a three-dimensional state space whose dynamics is governed by the following set of differential equation:

$$\frac{dz_1}{dt} = \sigma(z_2 - z_1) \tag{16}$$

$$\frac{dz_2}{dt} = \rho z_1 - z_2 - z_1 z_3 \tag{17}$$

$$\frac{dz_3}{dt} = z_1 z_2 - \beta z_3 \tag{18}$$

The parameter vectors $[\sigma\;\; \rho\;\; \beta]$ for the processes were chosen to be $[3\;\; 26.5\;\; 1]$ and $[4\;\; 30\;\; 1]$. The time evolution of the state for the first Lorenz process is shown in figure 3.

The ability to find the underlying factors which have generated the observations was already demonstrated in [4]. To make the problem more difficult, one dimension of each of the three elementary processes was hidden. As the original process has a total of eight states, five states are left from which the data was generated nonlinearly. In [4], some of the experiments used a random MLP network with inverse hyperbolic sine activation functions to generate ten-dimensional observations from five inputs. The same MLP network was used here. Like in [4], the data set consisted of 1000 observation.

Figure 4 shows the eight states of the underlying dynamical system, the five projections made from them and the noisy observations obtained from the projections by the random MLP network. The standard deviation of the observation noise was 0.1 while the standard deviation of the signal was normalised to unity.

4.2 Finding the underlying process

The NDFA algorithm was used for learning a dynamic model of the observations. Several different random initialisations of the MLP networks and structures of the model were tested. For the first 500 iterations, the concatenated vector $\mathbf{y}(t) = [\mathbf{x}(t-2)^T \; \dots \; \mathbf{x}(t+2)^T]^T$ was used instead of $\mathbf{x}(t)$ as the observation vectors. After that, $\mathbf{y}(t)$ was replaced by $\mathbf{x}(t)$ and the observation MLP was reduced accordingly. The cost function was found to be minimised by a model where there were ten factors and both the observation MLP network and the factor dynamics MLP network had one hidden layer of 30 neurons.

After 7500 iterations the model had learned a dynamic process which was able to represent the observations. The standard deviation of the observations was estimated to be 0.106 on the average which is in reasonably good agreement with the actual value of 0.1. In order to test the quality of the dynamic model learned by the algorithm, 1000 new values were predicted for the factors using the estimated mapping $\mathbf{g}$. The factors are shown in the upper part of figure 5.

The experiments with NLFA reported in [4] indicated that 7500 iterations were sufficient for learning the factors and the observation mapping. It turned out that more iterations are needed to fully learn the underlying dynamical process. Most of the learning was finished after 100,000 iterations, but some progress was observed even after 600,000 iterations. The simulation was not continued beyond that, however. In any case, the experiment confirms that ensemble learning is robust against overlearning, i.e., there is no need to control the complexity of the resulting mappings by early stopping of learning. The lower part of figure 5 shows the factors in the end of learning.

Visual inspection of the plots in figure 5 confirms that the NDFA algorithm has been able to capture the characteristics of the dynamics of the data-generating process. It also shows that only nine out of ten factors are actually used in the end. However, it is difficult to compare the estimated dynamics with the original by looking only at the predicted factors $\mathbf{s}(t)$. This is because the model learned by the NDFA uses a state representation which differs from the original.

Two processes can be considered equivalent if their state representations differ only by an invertible nonlinear transformation. As the original underlying states of the process are known, it is possible to examine the dynamics in the original state space. An MLP network was used for finding the mapping from the learned ten-dimensional factors to the original eight-dimensional states. The mapping was then used for visualising the dynamics in the original state space.

Figure 6 shows the reconstruction of the original states made from the predicted states $\mathbf{s}(t)$. First of all, it is evident that the factors contain all the required information about the state of the underlying process because the reconstructions are quite good for $1 \leq t \leq 1000$ even after 7500 iterations. Initially the dynamics is not modelled accurately enough to simulate the long term behaviour of the process, but in the end, the dynamics of all three underlying subprocesses are captured.

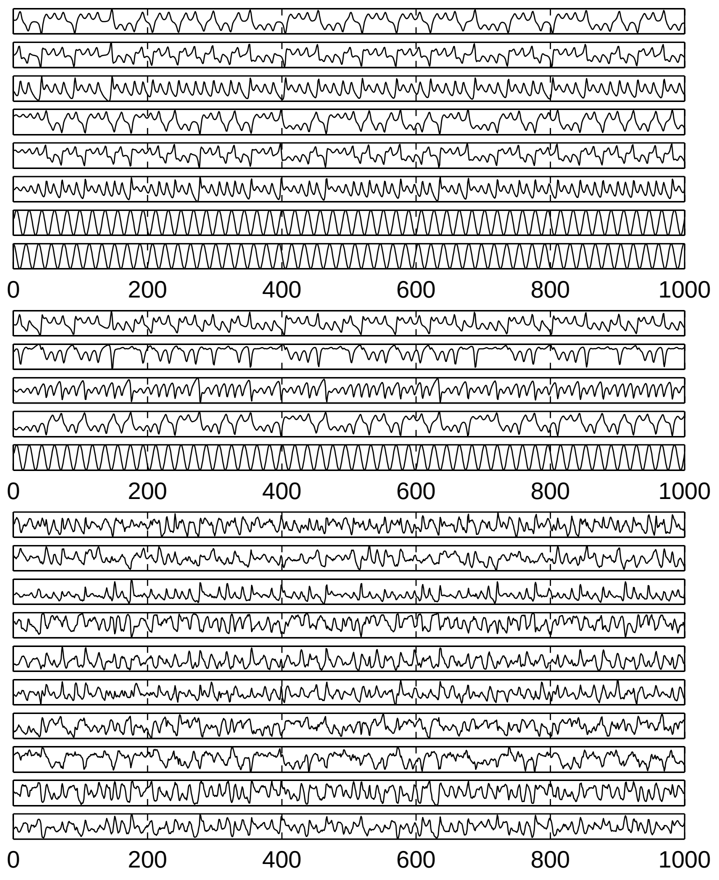

Figure 4: Each plot shows a time series. The eight plots on the top show the eight states on which the dynamics of the underlying process is defined. The process is composed of two Lorenz processes, each of which has three states, and a harmonic oscillator which has two states. Five projections from the eight states are used for generating the observations. These are shown in the middle. The ten plots on the bottom of the figure are the noisy observations which are obtained by instantaneous nonlinear mapping from the five projections.

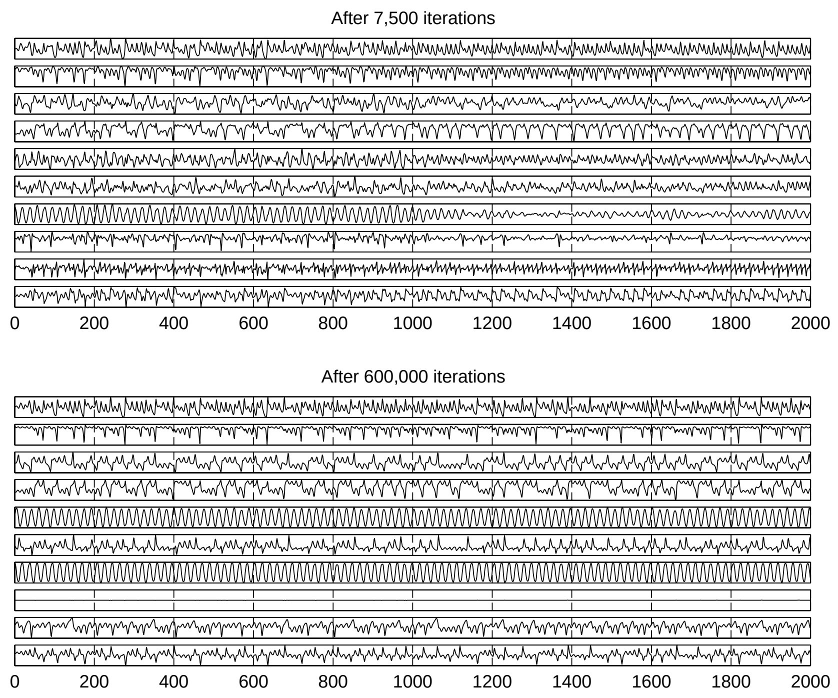

Figure 5: The ten plots on the top show the factors $\mathbf{s}(t)$ after 7500 iterations and the ten plots on the bottom show the factors after 600,000 iterations. The first 1000 values have been estimated based on the observations and the following 1000 values have been predicted using $\mathbf{s}(t) = \mathbf{g}(\mathbf{s}(t-1))$, i.e., without the innovation process.

4.3 Prediction accuracy

An obvious way to assess the quality of the learned model is to see on how long term the predictions given by the model are accurate. It should be noted that since the Lorenz process is chaotic, it is numerically impossible to predict it exactly in the distant future. The best that can be hoped for is to capture the overall aspects of the long-term behaviour. Figure 6 confirms that this is accomplished by the NDFA algorithm.

For comparison, the prediction of the future observations was tested with a nonlinear auto-regressive (NAR) model

$$\mathbf{x}(t) = \mathbf{h}(\mathbf{x}(t-1), \mathbf{x}(t-2), \dots, \mathbf{x}(t-D)) + \mathbf{n}(t)\,. \tag{19}$$

Several strategies were tested and the best performance was given by an MLP network with 20 inputs and one hidden layer of 30 neurons. The inputs to the MLP network were

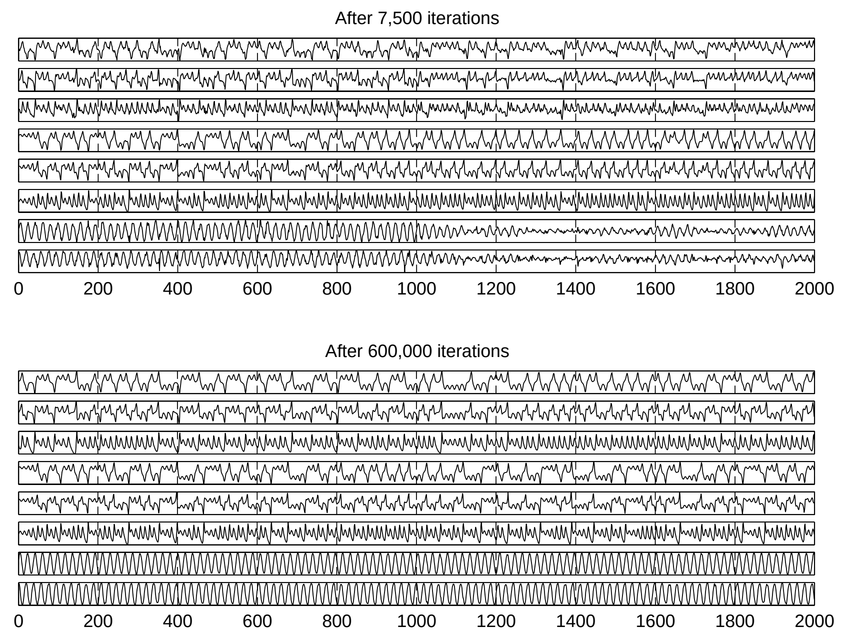

Figure 6: The original eight-dimensional state of the underlying process used for generating the observations is reconstructed from the predicted factors $\mathbf{s}(t)$ shown in figure 5. After 7500 iterations the model can follow the dynamics for a while and after 600,000 iterations the essential characteristics of the dynamics are captured with high fidelity as can be seen by comparing the original latent variables ($1 \leq t \leq 1000$) to the following 1000 predicted states.

the principal components extracted from the sequence of ten past observations, that is, D was ten. Figure 7 shows the performance of this NAR model. It is evident that the prediction of new observations is a challenging problem. Principal component analysis used in the model can extract features which are useful in predicting the future observations but clearly the features are not good enough for modelling the long-term behaviour of the observations.

Both the NAR model and the model used in NDFA algorithm contain noise process which can be taken into account in the prediction by Monte-Carlo simulation. Figure 8 shows the results obtained by 100 runs of Monte-Carlo simulation. The average predicted observations are compared to the actual continuation of the process. The figure shows the average cumulative squared prediction error. Since the variance of the noise on the observations is 0.01, the results can be considered perfect if the average squared prediction error is 0.01. The signal variance is 1, which gives the practical upper bound of the prediction accuracy. The figure confirms that the NDFA algorithm has been able to model

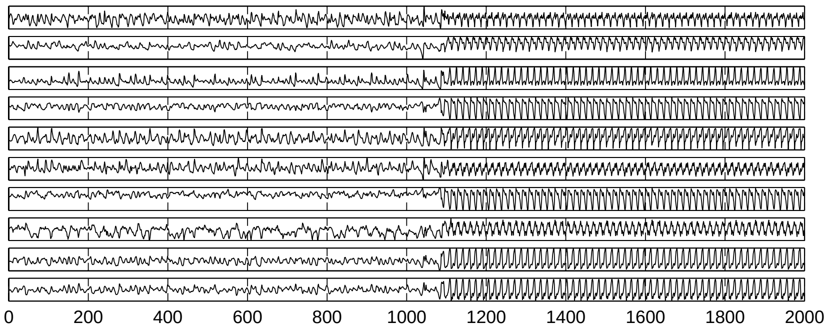

Figure 7: The ten plots show the original observations $\mathbf{x}(t)$ $(1 \leq t \leq 1000)$ and the result of prediction made by a nonlinear auto-regressive model $(t \geq 1001)$. The noise process was omitted like in the results shown in figures 5 and 6.

the dynamics of the process as the short-term average prediction error is close to the noise variance. Even the long-term prediction falls below the signal variance which indicates that at least some part of the process can be predicted in the more distant future. The steady progress made during learning is also evident.

Figures 9–12 show the results of the Monte-Carlo simulations for individual time series. Each plot shows the averages μ and the range $\mu \pm \sigma$ as obtained from the simulations. Here σ stands for the standard deviation. Figure 9 depicts the predicted continuation of the factors $\mathbf{s}(t)$. In figure 10, these are mapped to the original state space while figures 11 and 12 show the predicted observations obtained by the NDFA algorithm and NAR model, respectively, together with the true continuation.

5 Discussion

5.1 Independent and weakly coupled processes

The experiments show that the NDFA algorithm is able to extract predictable factors from the observations. The 1000 observations given for the algorithm can span the ten dimensional factor space only sparsely which indicates that the MLP network must have been able to generalise very well. This is possible since the observations were generated by lower-dimensional independent processes and the dynamics of the whole system can thus be expressed as a sum of the individual simple dynamics. This type of a mapping is easy to model with an MLP network.

The innovation process has a Gaussian prior which means that there is a rotational degeneracy in the model as certain rotations map the diagonal Gaussian density onto another diagonal Gaussian density. The rotation can be absorbed in the linear mappings of the MLP networks thus resulting in an equivalent model. However, the factors extracted by the algorithm are clearly very close to the original undelying time series used in generating the data. Each factor can be attributed to one process and none of the factors is a mixture

of the states of different elementary processes as would be expected if the algorithm would randomly settle to one of the equivalent rotations of the factor space.

The reason is that the approximation of the posterior probability of the factors cannot represent all the degenerate solutions equally well. The approximation assumes each factor to be independent of the other factors given the observations. The dynamic mapping induces posterior correlations which violate this assumption but the learning algorithm finds the solution which is closest to the assumption. It turns out that this is achieved by separating the underlying processes, thus yielding sparse dynamic couplings between different factors.

5.2 Future directions

In this work, adaptation of the approximation of the posterior density was done in batches, i.e., all the observations were processed before adaptation. It would be straight-forward to derive an on-line version of learning, however. For each new sample the new posterior approximation $q(\boldsymbol{\theta}|\mathbf{X}_{t+1})$ would be adapted by minimising the misfit between the new approximation and $p(\mathbf{x}(t+1)|\boldsymbol{\theta})q(\boldsymbol{\theta}|\mathbf{X}_t)/p(\mathbf{x}(t+1)|\mathbf{X}_t)$. Here $\boldsymbol{\theta}$ denotes all the unknown variables of the model, including factors, parameters of the mappings and hyperparameters. This type of on-line learning would be essentially a version of Kalman filtering [3].

The nonlinear dynamic model used here is universal in the sense that in principle any data generating process can be modelled by it with any given precision. The significance of this property is mostly theoretical since in practice the amount of observations limit the complexity of models which can be reliably learned from the data. The generalisation ability of the models can be improved by prior knowledge about the model structure. This should also make it easier to identify the factors with physical quantities of the observed system.

Like the NLFA algorithm, the proposed NDFA algorithm scales quadratically as a function of the dimension of factor space and several times larger factor spaces than in the experiments reported here are thus computationally feasible. However, with high-dimensional nonlinear models the interpretation of the results can be rather difficult. Due to the approximation made in the posterior, the algorithm was shown to extract factors each of which can be identified with one of the underlying independent processes. This should happen more reliably if the prior model would also promote sparse temporal couplings between the underlying factors, e.g., by using a sparse prior on the weight matrices of the MLP networks. Use of non-Gaussian model for the innovation process $\mathbf{m}(t)$ should also aid at the interpretation of the results in a similar way as non-Gaussian model of factors in linear factor analysis.

Another example of using prior knowledge could be implementation of memories with specific structure. The state representation can in principle learn any memory, but learning complex structures can be difficult in practice. It is also possible that some of the states or control signals of the underlying dynamical process are directly measured and they can then be included in the model as known external inputs.

6 Acknowledgements

This work was funded by the EU project BLISS. The author wishes to acknowledge the contribution of Mr. Juha Reunanen who assisted in doing many of the preliminary simulations which led to this work. The author would also like to thank Dr. J. P. Barnard for

suggesting the use of NLFA for extracting the embedded state of a dynamical system.

References

[1] T. Briegel and V. Tresp. Fisher scoring and a mixture of modes approach for approximate inference and learning in nonlinear state space models. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 403–409, Denver, Colorado, USA, Nov. 30–Dec. 5, 1998, 1999. The MIT Press.

[2] Z. Ghahramani and S. T. Roweis. Learning nonlinear dynamical systems using an EM algorithm. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 599–605, Denver, Colorado, USA, Nov. 30–Dec. 5, 1998, 1999. The MIT Press.

[3] M. S. Grewal and A. P. Andrews. *Kalman Filtering*. Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[4] 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.

[5] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20:130–141, 1963.

[6] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[7] S. T. Roweis and Z. Ghahramani. An EM algorithm for identification of nonlinear dynamical systems. In S. Haykin, editor, *Kalman Filtering and Neural Networks*. To appear.

[8] F. Takens. Detecting strange attractors in turbulence. In D. A. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence*, pages 366–381. Springer-Verlag, Berlin, 1981.

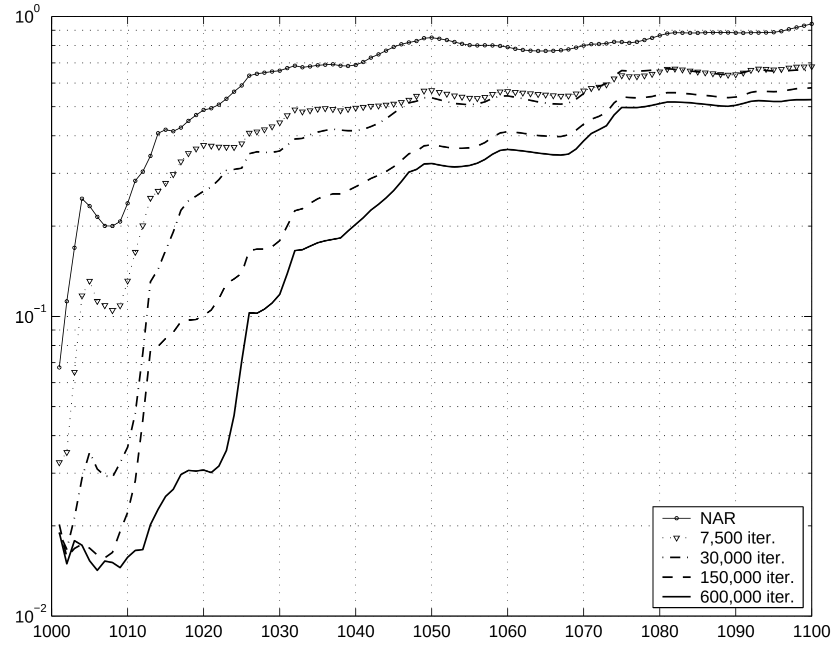

Figure 8: The average cumulative squared prediction error is computed for the predictions made using the NDFA algorithm after 7500 (dotted with triangles), 30,000 (dash-dotted), 150,000 (dashed) and and 600,000 iterations (solid) as well as by a nonlinear auto-regressive model (solid with dots). The predictions are based on 100 Monte-Carlo simulations of the estimated dynamics. Taking into consideration the observation noise whose variance is 0.01, the prediction obtained by the NDFA algorithm in the end of learning is excellent up to $t = 1010$ and fairly good up to $t = 1022$. The NAR model is quite inaccurate already after $t \geq 1003$.

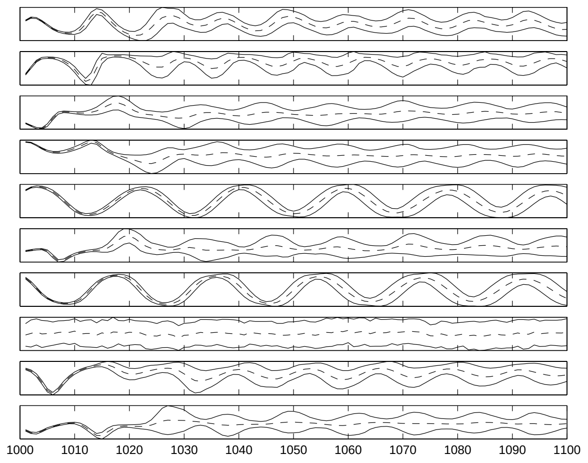

Figure 9: The factors $\mathbf{s}(t)$ have been predicted by Monte-Carlo simulations. The mean μ (dashed) and range $\mu \pm \sigma$ (solid) obtained from the simulation runs are shown.

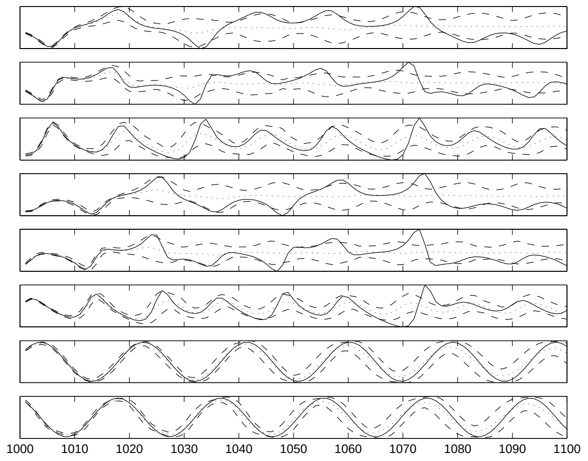

Figure 10: The factors in figure 9 are mapped on the original state space. The mean (dotted) and range (dashed) together with the true continuation (solid) are shown. Due to the chaotic nature of the Lorenz system, the state cannot be predicted for distant future. Notice, however, how the model has been able to predict the timing of the oscillations for the first and fourth states although the sign is uncertain.

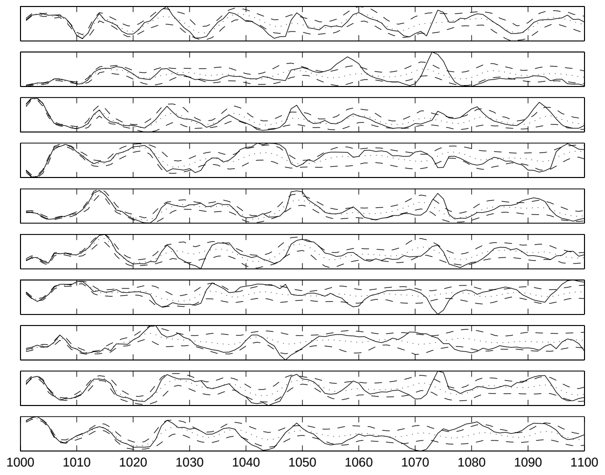

Figure 11: The factors in figure 9 are projected onto the observations by the estimated mapping $\mathbf{f}$. The mean (dotted) and range (dashed) together with the true continuation (solid) are shown.

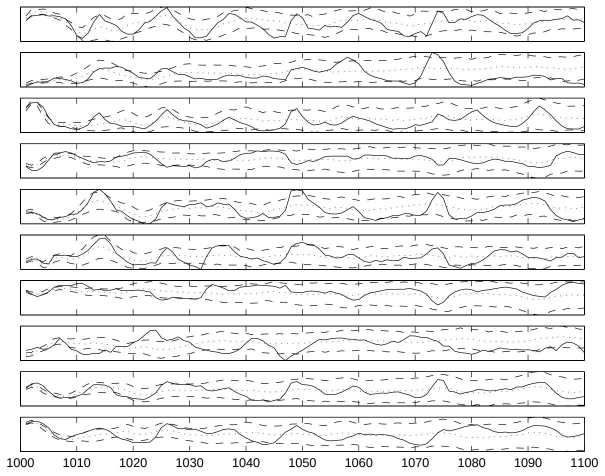

Figure 12: Monte-Carlo simulations with the NAR model have been used for predicting the observations. The mean (dotted) and range (dashed) together with the true continuation (solid) are shown.