HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Engineering
Physics and Mathematics

Antti Honkela

Nonlinear Switching State-Space Models

Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology

Espoo, May 29, 2001

Supervisor: Professor Juha Karhunen
Instructor: Harri Valpola, D.Sc. (Tech.)

TEKNILLINEN KORKEAKOULU	DIPLOMITYÖN TIIVISTELMÄ
TEKNILLISEN FYSIIKAN JA MATEMATIIKAN OSASTO	

Tekijä:	Antti Honkela
Osasto:	Teknillisen fysiikan ja matematiikan osasto
Pääaine:	Matematiikka
Sivuaine:	Informaatiotekniikka
Työn nimi:	Epälineaariset vaihtuvat tila-avaruusmallit
Title in English:	Nonlinear Switching State-Space Models
Professuurin koodi ja nimi:	Tik-61 Informaatiotekniikka
Työn valvoja:	Prof. Juha Karhunen
Työn ohjaaja:	TkT Harri Valpola

Tiivistelmä:

Epälineaarinen vaihtuva tila-avaruusmalli (vaihtuva NSSM) on kahden dynaamisen mallin yhdistelmä. Epälineaarinen tila-avaruusmalli (NSSM) on jatkuva ja kätketty Markov-malli (HMM) diskreetti. Vaihtuvassa mallissa NSSM mallittaa datan lyhyen aikavälin dynamiikkaa. HMM kuvaa pidempiaikaisia muutoksia ja ohjaa NSSM:a.

Tässä työssä kehitetään vaihtuva NSSM ja oppimisalgoritmi sen parametreille. Oppimisalgoritmi perustuu bayesiläiseen ensemble-oppimiseen, jossa todellista posteriorijakaumaa approksimoidaan helpommin käsiteltävällä jakaumalla. Sovitus tehdään todennäköisyysmassan perusteella ylioppimisen välttämiseksi. Algoritmin toteutus perustuu TkT Harri Valpolan aiempaan NSSM-algoritmiin. Se käyttää monikerrosperceptron -verkkoja NSSM:n epälineaaristen kuvausten mallittamiseen.

NSSM-algoritmin laskennallinen vaativuus rajoittaa vaihtuvan mallin rakennetta. Vain yhden dynaamisen mallin käyttö on mahdollista. Tällöin HMM:a käytetään vain NSSM:n ennustusvirheiden mallittamiseen. Tämä lähestymistapa on laskennallisesti kevyt mutta hyödyntää HMM:a vain vähän.

Algoritmin toimivuutta kokeillaan todellisella puhedatalla. Vaihtuva NSSM osoittautuu paremmaksi datan mallittamisessa kuin muut yleiset mallit. Työssä näytetään myös, kuinka algoritmi pystyy järkevästi segmentoimaan puhetta erillisiksi foneemeiksi, kun ainoastaan foneemien oikea järjestys tunnetaan etukäteen.

Sivumäärä: 93 Avainsanat: vaihtuva malli, hybridi, epälineaarinen tila-avaruusmalli, kätketty Markov-malli, ensemble-oppiminen

Täytetään osastolla

Hyväksytty: Kirjasto:

HELSINKI UNIVERSITY OF TECHNOLOGY ABSTRACT OF MASTER'S THESIS
DEPARTMENT OF ENGINEERING PHYSICS AND MATHEMATICS

Author:	Antti Honkela
Department:	Department of Engineering Physics and Mathematics
Major subject:	Mathematics
Minor subject:	Computer and Information Science
Title:	Nonlinear Switching State-Space Models
Title in Finnish:	Epälineaariset vaihtuvat tila-avaruusmallit
Chair:	Tik-61 Computer and Information Science
Supervisor:	Prof. Juha Karhunen
Instructor:	Harri Valpola, D.Sc. (Tech.)

Abstract:

The switching nonlinear state-space model (switching NSSM) is a combination of two dynamical models. The nonlinear state-space model (NSSM) is continuous by nature, whereas the hidden Markov model (HMM) is discrete. In the switching model, the NSSM is used to model the short-term dynamics of the data. The HMM describes the longer-term changes and controls the NSSM.

This thesis describes the development of a switching NSSM and a learning algorithm for its parameters. The learning algorithm is based on Bayesian ensemble learning, in which the true posterior distribution is approximated with a tractable ensemble. The approximation is fitted to the probability mass to avoid overlearning. The implementation is based on an earlier NSSM by Dr. Harri Valpola. It uses multilayer perceptron networks to model the nonlinear functions of the NSSM.

The computational complexity of the NSSM algorithm sets serious limitations for the switching model. Only one dynamical model can be used. Hence, the HMM is only used to model the prediction errors of the NSSM. This approach is computationally efficient but makes little use of the HMM.

The algorithm is tested with real-world speech data. The switching NSSM is found to be better in modelling the data than other standard models. It is also demonstrated how the algorithm can find a reasonable segmentation to different phonemes when only the correct sequence of phonemes is known in advance.

Number of pages: 93	Keywords:	switching model, hybrid model, nonlinear state-space model, hidden Markov model, ensemble learning
Department fills		
Approved:	Library code:	

Preface

This work has been funded by the European Commission research project BLISS. It has been done at the Laboratory of Computer and Information Science at the Helsinki University of Technology. It would not have been possible without the vast computational resources available at the laboratory. I am grateful to all my colleagues at the laboratory and the Neural Networks Research Centre for the wonderful environment they have created.

I wish to thank my supervisor, professor Juha Karhunen, for the opportunity to work at the lab and in the Bayesian research group. I am grateful to my instructor, Dr. Harri Valpola, for suggesting the subject of my thesis and giving a lot of technical advice on the way. They both helped in making the presentation of the work much more clear. I also wish to thank professor Timo Eirola for sharing his expertise on dynamical systems.

I am grateful to Mr. Vesa Siivola for his help in using the speech data in the experiments. Mr. Tapani Raiko and Mr. Jaakko Peltonen gave many useful suggestions in the numerous discussions we had.

Finally I wish to thank Maija for all her support during my project.

Otaniemi, May 29, 2001

Antti Honkela

Contents

List of abbreviations

CDHMM	Continuous density hidden Markov model
EM	Expectation maximisation
HMM	Hidden Markov model
ICA	Independent component analysis
MAP	Maximum a posteriori
MDL	Minimum description length
ML	Maximum likelihood
MLP	Multilayer perceptron (network)
NFA	Nonlinear factor analysis
NSSM	Nonlinear state-space model
PCA	Principal component analysis
pdf	Probability density function
RBF	Radial basis function (network)
SSM	State-space model

List of symbols

$\mathbf{A} = (a_{ij})$	Transition probability matrix of a Markov chain or a hidden Markov model
$\mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}$	The weight matrices and bias vectors of the MLP network $\mathbf{f}$
$\mathbf{C}, \mathbf{D}, \mathbf{c}, \mathbf{d}$	The weight matrices and bias vectors of the MLP network $\mathbf{g}$
$D(q(\theta)\|\|p(\theta))$	The Kullback–Leibler divergence between $q(\theta)$ and $p(\theta)$
$\mathrm{diag}[\mathbf{x}]$	A diagonal matrix with the elements of the vector $\mathbf{x}$ on the diagonal
$\mathrm{Dirichlet}(\mathbf{p};\ \mathbf{u})$	Dirichlet distribution for variable $\mathbf{p}$ with parameters $\mathbf{u}$
$\mathrm{E}[x]$	The expectation or mean of x
$\mathbf{f}$	Mapping from latent space to the observations or an MLP network modelling such a mapping
$\mathbf{g}$	Mapping modelling the continuous dynamics of an NSSM or an MLP network modelling such a mapping
$h(\mathbf{s})$	A measurement function $h : M \to \mathbb{R}$
$\mathcal{H}_i$	A model, hypothesis
M_t	Discrete hidden state or HMM state at time instant t
$\boldsymbol{M}$	The set of discrete hidden states or HMM states
$N(\mu, \sigma^2)$	Gaussian or normal distribution with parameters μ and σ^2
$N(x;\ \mu, \sigma^2)$	As $N(\mu, \sigma^2)$ but for variable x
$p(x)$	The probability of event x, or the probability density function evaluated at point x
$q(x)$	Approximating probability density function used in ensemble learning
$\mathbf{s}(t)$	Continuous hidden states or sources at time instant t
$s_k(t)$	The kth component of vector $\mathbf{s}(t)$
$\overline{s}_k(t)$	The estimated posterior mean of $s_k(t)$

$\mathring{s}_k(t)$	The estimated conditional posterior variance of $s_k(t)$ given $s_k(t-1)$
$\breve{s}_k(t-1,t)$	The estimated posterior linear dependence between $s_k(t-1)$ and $s_k(t)$
$\boldsymbol{S}$	The set of continuous hidden states or source values
$\mathrm{Var}[x]$	The variance of x
$\mathbf{x}(t)$	A sample of observed data
$\boldsymbol{X}$	The set of observed data
θ_i, θ	A scalar parameter of the model
$\overline{\theta}_i$	The estimated posterior mean of parameter θ_i
$\widetilde{\theta}_i$	The estimated posterior variance of parameter θ_i
$\boldsymbol{\theta}$	The set of all the parameters of the model
$\boldsymbol{\pi} = (\pi_i)$	Initial distribution vector of a Markov chain or a hidden Markov model
$\phi^t(\mathbf{x})$	The flow of a differential equation

Chapter 1

Introduction

1.1 Problem setting

Most data analysis methods are based on developing a *model* that could be used to recreate the studied data set. Speech recognition systems, for example, are often built around a model that could in principle be used as a speech generator. The success of the recogniser depends heavily on how well the generator can generate realistic speech data.

The speech generators used by most modern speech recognition systems are based on the *hidden Markov model* (HMM). The HMM is a *discrete* model. It has a finite number of different *internal states* that produce different kind of output. Typically there are a couple of states for each phoneme or a pair of phonemes. The whole dynamical process of producing speech is thus modelled by discrete transitions between the states corresponding to the different phonemes.

The model of human speech implied by the HMM is not a very realistic one. The dynamics of the mouth and the vocal cord used to produce the speech are continuous. The discrete model is only a very crude approximation of the "true" model. A more realistic approach would be to model the data with a continuous model. The process of producing speech is clearly nonlinear and this should be reflected by its model. A good candidate for the task is the *nonlinear state-space model* (NSSM). The NSSM can be described as the continuous counterpart of the HMM. The problem with models like the NSSM is that they concentrate on modelling the short-term structure of the data. Therefore they are not as such very well suited for speech recognition.

There are speech recognition systems that try to get the best of the both worlds by combining the two different kinds of models into one hybrid structure. Such systems have performed well in several difficult real world problems but they are often rather specialised. The training algorithms for such models are usually based on some heuristic measures rather than on generally accepted mathematical principles.

In this work, a hybrid model structure that combines the HMM with another dynamical model, the continuous NSSM, is studied. The resulting model is called the *switching nonlinear state-space model* (switching NSSM). The resulting hybrid model has the power of a continuous NSSM to model the short-term dynamics of the data. However, above the NSSM there is still the familiar HMM to divide the data to different discrete states corresponding, for example, to the different phonemes.

1.2 Aim of the thesis

The aim of this thesis has been to develop a *Bayesian* formulation for the switching NSSM and a *learning algorithm* for the parameters of the model. The term learning algorithm in this context means a procedure for optimising the parameters of the model to best describe the given data. The learning algorithm is based on the approximation method called *ensemble learning.* It provides a principled approach for global optimisation of the performance of the model. Similar switching models exist but they use only linear state-space models (SSMs).

In practice the switching model has been implemented by extending the existing NSSM model and learning algorithm developed by Dr. Harri Valpola [58, 60]. The performance of the developed model has been verified by applying it to a data set of Finnish speech in two different experiments. In the first experiment, the switching NSSM has been compared with plain HMM, standard NSSM without switching and a static nonlinear factor analysis model which completely ignores the temporal structure of the data. In the second experiment, patches of speech with known annotation, i.e. the sequence of phonemes in the word, have been used. The correct segmentation of the word to the individual phonemes was, however, not known, and must be learnt by the model.

Even though the development of the model has been motivated here by speech recognition examples, the purpose of this thesis has not been to develop a working speech recognition system. Such a system could probably be developed by extending the work presented here.

1.3 Structure of the thesis

The thesis begins with a review of different methods of time series modelling in Chapters 2–4. Chapter 2 presents a mathematical point of view to the subject. These ideas form the foundation for what follows but most of the results are not used directly. Statistical methods for learning the parameters of the different models are presented in Chapter 3. Chapter 4 introduces the building blocks of the switching NSSM and discusses previous work on HMMs and SSMs, as well as some of their combinations.

The second half of the thesis (Chapters 5–7) consists of the development of the switching NSSM and experimental verification of its operation. In Chapter 5, the exact structures of the parts of the switching model, the HMM and the NSSM, are defined. It is also described how the two models are combined. The ensemble learning based learning algorithms for all the models of the previous chapter are derived in Chapter 6. A series of experiments was conducted to verify the performance of the model. Chapter 7 discusses these experiments and their results. Finally the results of the thesis are summarised in Chapter 8.

The thesis also has two appendices. Appendix A presents two important probability distributions, the Gaussian and the Dirichlet distribution, and some of their most important properties. Appendix B contains a derivation of a result needed in the learning algorithm in Chapter 6.

1.4 Contributions of the thesis

The first part of the thesis consists of a literature review of important background knowledge for developing the switching NSSM. The model is presented in Chapter 5. The continuous density HMM in Section 5.1 has been developed by the author although it is heavily based on earlier work by Dr. David MacKay [39]. A Bayesian NSSM by Dr. Harri Valpola is presented in Section 5.2. The modifications needed to add switching to the NSSM model, as presented in Section 5.3, have been developed by the author. The same division applies to the learning algorithms for the models, found in Chapter 6.

The developed learning algorithm has been tested extensively. The author has implemented the switching NSSM learning algorithm under Matlab. The code is based on an implementation of the NSSM, which is in turn based on an implementation of the nonlinear factor analysis (NFA) algorithm. Dr. Harri Valpola wrote the extensions from NFA to NSSM. The author has written most of the rest of the code over a period of two years. All the experiments reported in Chapter 7 and the HMM implementation used in the comparisons have been done by the author.

Chapter 2

Mathematical preliminaries of time series modelling

As this thesis deals with modelling time series data, the first chapter will discuss mathematical background on the subject. In Section 2.1, a brief introduction to the theory of dynamical systems and some useful tools for solving the related *inverse problem*, i.e. finding a suitable dynamical system to model a given time series, is presented. Section 2.2 discusses some of the practical consequences of the results. The concepts presented in this chapter form the basis for the rest of the thesis. Most of them will, however, not be used directly.

2.1 Theory

2.1.1 Dynamical systems

The theory of dynamical systems is the basic mathematical tool for analysing time series. This section presents a brief introduction to the basic concepts. For a more extensive treatment, see for example [1].

The general form for an autonomous discrete-time dynamical system is the *map*

$$\mathbf{x}_{n+1} = \mathbf{f}(\mathbf{x}_n) \tag{2.1}$$

where $\mathbf{x}_n, \mathbf{x}_{n+1} \in \mathbb{R}^n$ and $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a *diffeomorphism*, i.e. a smooth mapping with a smooth inverse. It is important that the mapping $\mathbf{f}$ is independent of time, meaning that it only depends on the argument point $\mathbf{x}_n$. Such mappings are often generated by *flows* of autonomous differential equations.

For a general autonomous differential equation

$$\mathbf{x}'(t) = \mathbf{f}(\mathbf{x}(t)), \tag{2.2}$$

we define the *flow* by [1]

$$\phi^t(\mathbf{x}_0) := \mathbf{x}(t) \tag{2.3}$$

where $\mathbf{x}(t)$ is the unique solution of Equation (2.2) with the initial condition $\mathbf{x}(0) = \mathbf{x}_0$, evaluated at time t. The function $\mathbf{f}$ in Equation (2.2) is called the *vector field* corresponding to the flow ϕ.

Setting $g(\mathbf{x}) := \phi^\tau(\mathbf{x})$, where $\tau > 0$, gives an autonomous discrete-time dynamical system like in Equation (2.1). The discrete system defined in this way samples the values of the continuous system at constant intervals τ. Thus it is a discretisation of the continuous system.

2.1.2 Linear systems

Let us assume that the mapping $\mathbf{f}$ in Equation (2.1) is linear, i.e.

$$\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n. \tag{2.4}$$

Iterating the system for a given initial vector $\mathbf{x}_0$ leads to a sequence

$$\mathbf{x}_n = \mathbf{A}^n\mathbf{x}_0. \tag{2.5}$$

The possible courses of evolution of such sequences can be characterized by the eigenvalues of the matrix $\mathbf{A}$. If there are eigenvalues that are greater than one in absolute value, almost all of the orbits will diverge to infinity. If all the eigenvalues are less than one in absolute value, the orbits will rapidly converge to the origin. Complex eigenvalues with absolute value of unity will lead to closed circular or elliptic orbits [1].

An affine map with $\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n + \mathbf{b}$ behaves in essentially the same way. This shows that the autonomous linear system is too simple to describe any interesting dynamical phenomena, because in practice the only stable linear systems converge exponentially to a constant value.

2.1.3 Nonlinear systems and chaos

While the possible dynamics of linear systems are rather restricted, even very simple nonlinear systems can have very complex dynamical behaviour. Nonlinearity is closely associated with chaos, even though not all nonlinear mappings produce chaotic dynamics [49].

A dynamical system is chaotic if its future behaviour is very sensitive to the initial conditions. In such systems two orbits starting close to each other will rather soon behave very differently. This makes any long term prediction of the evolution of the system impossible. Even for a deterministic system that is perfectly known, the initial conditions would have to be known arbitrarily precisely, which is of course impossible in practice. There is, however, great variation in how far ahead different systems can be predicted.

A classical example of a chaotic system is the weather conditions in the atmosphere of the Earth. It is said that a butterfly flapping its wings in the Caribbean can cause a great storm in Europe a few weeks later. Whether this is actually true or not, it gives a good example on the futility of trying to model a chaotic system and predict its evolution for long periods of time.

Even though long term prediction of a chaotic system is impossible, it is still often possible to predict statistical features of its behaviour. While modelling weather has proven troublesome, modelling the general long term behaviour, the climate, is possible. It is also possible to find certain invariant features of chaotic systems that provide qualitative description of the system.

2.1.4 Tools for time series analysis

In a typical mathematical model [9], a time series $(x(t_1), \ldots, x(t_n))$ is generated by the flow of a smooth dynamical system on a d-dimensional smooth manifold M:

$$\mathbf{s}(t) = \phi^t(\mathbf{s}(0)). \tag{2.6}$$

The original d-dimensional states of the system cannot be observed directly. Instead, the observations consist of the possibly noisy values $x(t)$ of a one-dimensional measurement function h which are related to the original states by

$$x(t) = h(\mathbf{s}(t)) + n(t) \tag{2.7}$$

where $n(t)$ denotes some noise process corrupting the observations. We shall for now assume that the observations are noiseless, i.e. $n(t) = 0$, and deal with the noisy case later on.

Reconstructing the state-space

The first problem in modelling a time series like the one described by Equations (2.6) and (2.7) is to try to reconstruct the original state-space or its equivalent, i.e. to find the structure of the manifold M.

Two spaces are topologically equivalent if there exists a continuous mapping with a continuous inverse between them. In this case it is sufficient that the equivalent structure is a part of a larger entity, the rest can easily be ignored. Thus the interesting concept is *embedding*, which is defined as follows.

Definition 1. *A function $f : X \to Y$ is an* embedding *if it is a continuous mapping with a continuous inverse $f^{-1} : f(X) \to X$ from its range to its domain.*

Whitney showed in 1936 [62] that any d-dimensional manifold can be embedded into $\mathbb{R}^{2d+1}$. The theorem can be extended to show that with a proper definition of *almost all* for an infinite-dimensional function space, almost all smooth mappings from given d-dimensional manifold to $\mathbb{R}^{2d+1}$ are embeddings [53].

Having only a single time series produced by Equation (2.7), how does one get those $2d + 1$ different coordinates? This problem can usually be solved by introducing so called *delay coordinates* [53].

Definition 2. *Let ϕ be a flow on a manifold M, $\tau > 0$, and $h : M \to \mathbb{R}$ a smooth measurement function. The* delay coordinate map *with embedding delay τ, $F(h, \phi, \tau) : M \to \mathbb{R}^n$ is defined by*

$$\mathbf{s} \mapsto \left(h(\mathbf{s}), h(\phi^{-\tau}(\mathbf{s})), h(\phi^{-2\tau}(\mathbf{s})), \ldots, h(\phi^{-(n-1)\tau}(\mathbf{s}))\right).$$

Takens proved in 1980 [55] that such mappings can indeed be used to reconstruct the state-space of the original dynamical system. This result is known as *Takens' embedding theorem.*

Theorem 3 (Takens). *Let M be a compact manifold of dimension d. For triples (f, h, τ), where f is a smooth vector field on M with flow ϕ, $h : M \to \mathbb{R}$ a smooth measurement function and the embedding delay $\tau > 0$, it is a* generic property *that the delay coordinate map $F(h, \phi, \tau) : M \to \mathbb{R}^{2d+1}$ is an* embedding.

Takens' theorem states that in the general case, the dynamics of the system recovered by delay coordinate embedding are the same as the dynamics of the original system. The exact mathematical definition of this "in general" is, however, somewhat more complicated [46].

Definition 4. *A subset $U \subset X$ of a topological space is* residual *if it contains a countable intersection of open dense subsets. A property is called* generic *if it holds in a residual set.*

According to Baire's theorem, a residual set cannot be empty. Unfortunately that is about all that can be said about it. Even in Euclidean spaces, a residual set can be of arbitrarily small measure. With a proper definition for *almost all* in infinite-dimensional spaces and slightly different assumptions, Sauer et al. showed in 1991 that the claim of Takens' theorem actually applies almost always [53].

2.2 Practical considerations

According to Takens' embedding theorem, almost all dynamical systems can be reconstructed from just one *noiseless* observation sequence. Unfortunately such observations of real life systems are very hard to get. There is always some measurement noise and even if that could be eliminated, the results must be stored into a computer with finite precision to do any practical calculations with them.

In practice the observation noise can be a serious hinder to the reconstruction. Low dimensional observations from a high dimensional chaotic system with high noise level can easily become indistinguishable from a random time series.

2.2.1 Delay coordinates in practice

Even though Takens' theorem does not give any guarantees of the success of the embedding procedure in the noisy case, the delay coordinate embedding has been found useful in practice and is used widely [22]. In the noisy case, having more than one-dimensional measurements of the same process can help very much in the reconstruction even though Takens' theorem achieves the goal with just a single time series.

The embedding dimension must also be chosen carefully. Too low dimensionality may cause problems with noise amplification but using too high dimensionality will inflict other problems and is computationally expensive [9]. Assuming there is access to the original continuous measurement stream there is also another free parameter in the procedure, namely choosing the embedding delay τ. Takens' theorem applies for almost all delays, at least as long as an infinitely long series of noiseless observations is used. According to Haykin and Principe [22] the delay should be chosen to be long enough for the consecutive observations to be essentially, but not too, independent. In practice a good value can often be found at the first minimum of the mutual information between the consecutive samples.

2.2.2 Prediction algorithms

Assume we are analysing the scalar time series $x(1), \ldots, x(T)$. Using the time-delay embedding with delay d transforms the series to vectors of the form $\mathbf{y}(t) = (x(t-d+1), \ldots, x(t))$. Prediction in these coordinates corresponds to predicting the next sample $x(t+1)$ from the d previous ones, as all the other components of $\mathbf{y}(t+1)$ are directly available in $\mathbf{y}(t)$. Thus the problem reduces to finding a predictor of the form

$$x(t+1) = f(\mathbf{y}(t)). \tag{2.8}$$

Using a simple linear function for f leads to a linear *auto-regressive* (AR) model [20]. There are also several generalisations of the same model leading to other algorithms for the same purpose.

Farmer and Sidorowich [14] propose a locally linear predictor in which there are several linear predictors for different areas of the embedded state-space. The achieved performance is comparable with the global linear predictor for small embedding dimensionalities but as the embedding dimension grows, the local method clearly outperforms the global one. The problem with the locally linear predictor is that it is not continuous.

Casdagli [8] presents a review of several different predictors including a globally linear predictor, a locally linear predictor and a global nonlinear predictor using a *radial basis function* (RBF) network [21] as the nonlinearity. According to his results, the nonlinear methods are best for small data sets whereas the locally linear method of Farmer and Sidorowich gives the best results for large data sets.

For noisy data, the choice of coordinates can make a big difference on the success of the prediction algorithm. The embedding approach is by no means the only possible alternative, and as Casdagli et al. note in [9], there are often clearly better alternatives. There are many approaches in the field of neural computation where the choice is left to the algorithm. This corresponds to trying to blindly invert Equation (2.7). Such models are called *(nonlinear) state-space models* and they are studied in detail in Section 4.2.

Chapter 3

Bayesian methods for data analysis

Inclusion of the effects of noise into the model of a time series leads to the world of statistics — it is no longer possible to talk about exact events, only their probabilities.

The *Bayesian framework* offers the mathematically soundest basis for doing statistical work. In this chapter, a brief review of the most important results and tools of the field is presented.

Section 3.1 concentrates on the basic ideas of Bayesian statistics. Unfortunately, exact application of those methods is usually not possible. Therefore Section 3.2 discusses some practical approximation methods that allow getting reasonably good results with limited computational resources. The learning algorithms presented in this work are based on the approximation method called *ensemble learning*, which is presented in Section 3.3.

This chapter contains many formulas involving probabilities. The notation $p(x)$ is used for both probability of a discrete event x and the value of the *probability density function* (pdf) of a continuous variable at x, depending on what x is. All the theoretical results presented apply equally to both cases, at least when integration over a discrete variable is interpreted in the Lebesgue sense as summation.

Some authors use subscripts to separate different pdfs but here they are omitted to simplify the notation. All pdfs are identified only by the argument of p.

Two important probability distributions, the Gaussian or normal distribution and the Dirichlet distribution are presented in Appendix A. The notation $p(x) = N(x;\ \mu, \sigma^2)$ is used to denote that x is normally distributed with mean μ and variance σ^2.

3.1 Bayesian statistics

In the Bayesian probability theory, the probability of an event describes the observer's *degree of belief* on the occurrence of the event [36]. This allows evaluating, for instance, the probability that a certain parameter in a complex model lies on a certain fixed interval.

The Bayesian way of estimating the parameters of a given model focuses around the Bayes theorem. Given some data $\boldsymbol{X}$ and a model (or hypothesis) $\mathcal{H}$ for it that depends on a set of parameters $\boldsymbol{\theta}$, the Bayes theorem gives the posterior probability of the parameters

$$p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H}) = \frac{p(\boldsymbol{X}|\boldsymbol{\theta}, \mathcal{H})p(\boldsymbol{\theta}|\mathcal{H})}{p(\boldsymbol{X}|\mathcal{H})}. \tag{3.1}$$

In Equation (3.1), the term $p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H})$ is called the *posterior probability* of the parameters. It gives the probability of the parameters, when the data and the model are given. Therefore it contains all the information about the values of the parameters that can be extracted from the data. The term $p(\boldsymbol{X}|\boldsymbol{\theta}, \mathcal{H})$ is called the *likelihood* of the data. It is the probability of the data, when the model and its parameters are given and therefore it can usually be evaluated rather easily from the definition of the model. The term $p(\boldsymbol{\theta}|\mathcal{H})$ is the *prior probability* of the parameters. It must be chosen beforehand to reflect one's prior belief of the possible values of the parameters. The last term $p(\boldsymbol{X}|\mathcal{H})$ is called the *evidence* of the model $\mathcal{H}$. It can be written as

$$p(\boldsymbol{X}|\mathcal{H}) = \int p(\boldsymbol{X}|\boldsymbol{\theta}, \mathcal{H})p(\boldsymbol{\theta}|\mathcal{H})d\boldsymbol{\theta} \tag{3.2}$$

and it ensures that the right hand side of the equation is properly scaled. In any case it is just a constant that is independent of the values of the parameters and can thus be usually ignored when inferring the values of the parameters of the model. This way the Bayes theorem can be written in a more compact form

$$p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H}) \propto p(\boldsymbol{X}|\boldsymbol{\theta}, \mathcal{H})p(\boldsymbol{\theta}|\mathcal{H}). \tag{3.3}$$

The evidence is, however, very important when comparing different models.

The key idea in Bayesian statistics is to work with full distributions of parameters instead of single values. In calculations that require a value for a certain parameter, instead of choosing a single "best" value, one must use all the values and weight the results according to the posterior probabilities of the parameter values. This is called *marginalising* over the parameter.

3.1.1 Constructing probabilistic models

To use the techniques of probabilistic modelling, one must usually somehow specify the likelihood of the data given some parameters. Not all models are, however, probabilistic in nature and have naturally emerging likelihoods. Many models are defined as *generative models* by stating how the observed data could be *generated* from unknown parameters or *latent variables*.

For given data vectors $\mathbf{x}(t)$, a simple linear generative model could be written as

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) \tag{3.4}$$

where $\mathbf{A}$ is an unobserved transformation matrix and vectors $\mathbf{s}(t)$ are unobserved *hidden* or *latent variables*. In some application areas the latent variables are also called *sources* or *factors* [6, 35].

One way to turn such a generative model into a probabilistic model is to add a noise term to the Equation (3.4). This yields

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t) \tag{3.5}$$

where the noise can, for example, be assumed to be zero-mean and Gaussian as in

$$\mathbf{n}(t) \sim N(0, \mathbf{\Sigma_n}). \tag{3.6}$$

Here $\mathbf{\Sigma_n}$ denotes the covariance matrix of the noise and it is usually assumed to be diagonal to make the different components of the noise independent.

This implies a likelihood for the data given by

$$p(\mathbf{x}(t)|\mathbf{A}, \mathbf{s}(t), \mathbf{\Sigma_n}, \mathcal{H}) = N(\mathbf{x}(t);\ \mathbf{A}\mathbf{s}(t), \mathbf{\Sigma_n}). \tag{3.7}$$

For a complete probabilistic model one also needs priors for all the parameters of the model. For instance hierarchical models and conjugate priors can be useful mathematical tools here but in the end the priors should be chosen to represent true prior knowledge on the solution of the problem at hand.

3.1.2 Hierarchical models

Many models include groups of parameters that are somehow related or connected. This connection should be reflected in the prior chosen for them. *Hierarchical models* provide a useful tool for building priors for such groups. This is done by giving the parameters a common prior distribution which is parameterised with new higher level hyperparameters [16].

Such a group would typically include parameters that have a somehow similar status in the model. Hierarchical models are well suited for neural network related problems because such connected groups emerge naturally, like for example the different elements of a weight matrix.

The definitions of the components of the Bayesian nonlinear switching state-space model in Chapter 5 contain several examples of hierarchical priors.

3.1.3 Conjugate priors

For a given class of likelihood functions $p(\boldsymbol{X}|\boldsymbol{\theta})$, the class $\mathcal{P}$ of priors $p(\boldsymbol{\theta})$ is called a *conjugate* if the posterior $p(\boldsymbol{\theta}|\boldsymbol{X})$ is of the same class $\mathcal{P}$.

This is a very useful property if the class $\mathcal{P}$ consists of a set of probability densities with the same functional form. In such a case the posterior distribution will also have the same functional form. For instance the conjugate prior for the mean of a Gaussian distribution is Gaussian. In other words, if the prior of the mean and the functional form of the likelihood are Gaussian, the posterior of the mean will also be Gaussian [16].

3.2 Posterior approximations

Even though the Bayesian statistics gives the optimal method for performing statistical inference, the exact use of those tools is impossible for all but the simplest models. Even if the likelihood and prior can be evaluated to give an unnormalised posterior of Equation (3.3), the integral needed for the scaling term of Equation (3.2) is usually intractable. This makes analytical evaluation of the posterior impossible.

As the exact Bayesian inference is usually impossible, there are many algorithms and methods that approximate it. The simplest method is to approximate the posterior with a discrete distribution concentrated at the maximum of the posterior density given by Equation (3.3). This gives a single value for all the parameters. The method is called *maximum a posteriori* (MAP) estimation. It is closely related to the classical technique of *maximum likelihood* (ML) estimation, in which the contribution of the prior is ignored and only the likelihood term $p(\boldsymbol{X}|\boldsymbol{\theta}, \mathcal{H})$ is maximised [57].

The MAP estimate is troublesome because especially in high dimensional spaces, high probability density does not necessarily have anything to do with high probability mass, which is the quantity of interest. A narrow spike can have very high density, but because of its very small width, the actual probability of the studied parameter belonging to it is small. In high dimensional spaces the width of the mode is much more important than its height.

As an example, let us consider a simple linear model for data $\mathbf{x}(t)$

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) \tag{3.8}$$

where both $\mathbf{A}$ and $\mathbf{s}(t)$ are unknown. This is the generative model for standard PCA and ICA [27].

Assuming that both $\mathbf{A}$ and $\mathbf{s}(t)$ have unimodal prior distributions centered at the origin, the MAP solution will typically give very small values for $\mathbf{s}(t)$ and very large values for $\mathbf{A}$. This is because there are so many more parameters in $\mathbf{s}(t)$ than in $\mathbf{A}$ that it pays off to make the sources very close to their prior most probable value, even at the cost of $\mathbf{A}$ having huge values. Of course such a solution cannot make any sense, because the source values must be specified very precisely in order to describe the data. In simple linear models such behaviour can be suppressed by restricting the values of $\mathbf{A}$ suitably. In more complex models there usually is no way to restrict the values of the parameters and using better approximation methods is essential.

The same problem in a two dimensional case is illustrated in Figure 3.1.[1] The mean of the two dimensional distribution in the figure lies near the centre of the square where most of the probability mass is concentrated. The narrow spike has high density but it is not very massive. Using a gradient based algorithm to find the maximum of the distribution (MAP estimate) would inevitably lead to the top of the spike. The situation of the figure may not look that bad but the problem gets much worse when the dimensionality is higher.

[1]The basic idea of the figure is due to Barber and Bishop [2].

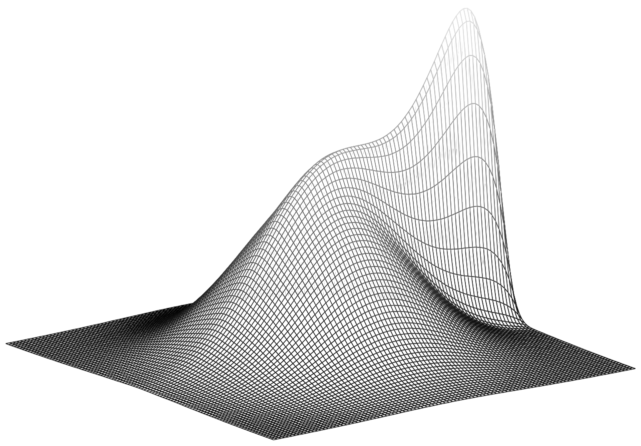


Figure 3.1: High probability density does not always imply high probability mass. The spike on the right has the highest density even though most of the mass is near the centre.

3.2.1 Model selection

According to the marginalisation principle, the correct way to compare different *models* in the Bayesian framework is to always use *all* the models, weighting their results by the respective posterior probabilities. This is a computationally demanding approach and it may be desirable to use only one model, even if it does not lead to optimal results. The rest of the section concentrates on finding suitable criteria for choosing just one "best" model.

Occam's razor is an old scientific principle which states that when trying to explain some phenomenon, the simplest model that can adequately explain it should be chosen. There is no point in choosing an overly complex model when even a much simpler one would do. A very complex model will be able to fit the given data almost perfectly but it will not be able to generalise very well. On the other hand, very simple models will not be able to describe the essential features of the data. One must make a compromise and choose a model with enough but not too much complexity [57, 10].

MacKay showed in [37] how this can be done in the Bayesian framework by evaluating and comparing model *evidences*. The evidence of a model $\mathcal{H}_i$ is defined as the probability $p(\boldsymbol{X}|\mathcal{H}_i)$ of the data given the model. This is just the scaling factor in the denominator of the Bayes theorem in Equation (3.1) and it can be evaluated as shown in Equation (3.2).

True Bayesian comparison of the models would require using Bayes theorem to get the posterior probability of the model as

$$p(\mathcal{H}_i|\boldsymbol{X}) \propto p(\boldsymbol{X}|\mathcal{H}_i)p(\mathcal{H}_i). \tag{3.9}$$

If, however, there is no reason to prefer one model over another, one can choose a uniform prior for all the models. This way the posterior probability of a model is directly proportional to the evidence. There is no need to use a prior on the models to penalise complex ones, the evidence will do that automatically.

Figure 3.2 shows how the model evidence can be used to choose the right model.[2] In the figure the horizontal axis ranges through all the possible data sets. The curves show the values of evidence for different models and different data sets. As the distributions $p(\boldsymbol{X}|\mathcal{H}_i)$ are all normalised, the area under each curve is equal.

A simple model like $\mathcal{H}_1$ can only describe a small range of possible data sets. It gives a high evidence for those but nothing for the rest. A very complex model like $\mathcal{H}_2$ can describe a much larger variety of data sets. Therefore it has to spread its predictions more thinly than model $\mathcal{H}_1$ and gives lower evidence for simple data sets. And for a data set like $\boldsymbol{X}_1$ lying there in the middle, both the extremes will lose to model $\mathcal{H}_3$ which is just good enough for that data and therefore just the model called for by Occam's razor.

After this point the explicit references to the model $\mathcal{H}$ in expressions for different probabilities are omitted. This is done purely to simplify the notation. It should be noted that in the Bayesian framework, all the probabilities are conditional to some assumptions because they are always subjective.

3.2.2 Stochastic approximations

The basic idea in stochastic approximations is to somehow get samples from the true posterior distribution. This is usually done by constructing a Markov chain for the model parameters $\boldsymbol{\theta}$ whose stationary distribution corresponds to the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{X})$. Simulation algorithms doing this are called *Markov chain Monte Carlo* (MCMC) methods.

The most important of such algorithms is the *Metropolis–Hastings* algorithm. To use it, one must be able to compute the unnormalised posterior of Equa-

[2]The idea of the figure is due to Dr. David MacKay [37].

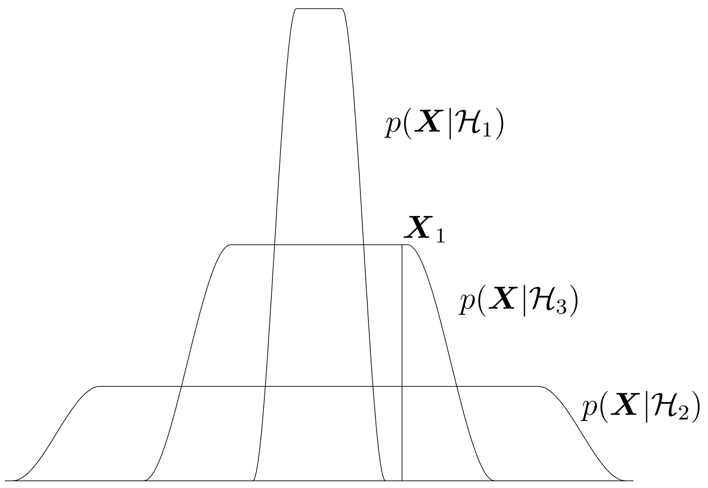

$p(\boldsymbol{X}|\mathcal{H}_1)$
$\boldsymbol{X}_1$
$p(\boldsymbol{X}|\mathcal{H}_3)$
$p(\boldsymbol{X}|\mathcal{H}_2)$

Figure 3.2: How model evidence embodies Occam's razor [37]. For given data set $\boldsymbol{X}_1$, the model with just right complexity will have the highest evidence.

tion (3.3). In addition one must specify a *jumping distribution* for the parameters. This can be almost any reasonable distribution that models possible transitions between different parameter values. The algorithm works by getting random samples from the jumping distribution and then either taking the suggested transition or rejecting it, depending on the ratio of the posterior probabilities of the two values in question. The normalising factor of the posterior is not needed as only the ratio of the posterior probabilities is needed [16].

Neal [44] discusses the use of MCMC methods for neural networks in detail. He also presents some modifications to the basic algorithms to improve their performance for large neural network models.

3.2.3 Laplace approximation

Laplace's method approximates the integral of a function $\int f(\mathbf{w})d\mathbf{w}$ by fitting a Gaussian at the maximum $\widehat{\mathbf{w}}$ of $f(\mathbf{w})$, and computing the volume under the Gaussian. The covariance of the fitted Gaussian is determined by the Hessian matrix of $\log f(\mathbf{w})$ at the maximum point $\widehat{\mathbf{w}}$ [40].

The same name is also used for the method of approximating the posterior distribution with a Gaussian centered at the *maximum a posteriori* estimate. This can be justified by the fact that under certain regularity conditions, the posterior distribution approaches Gaussian distribution as the number of samples grows [16].

Despite using a full distribution to approximate the posterior, Laplace's method still suffers from most of the problems of MAP estimation. Estimating the variances at the end does not help if the procedure has already lead to an area of low probability mass.

3.2.4 EM algorithm

The *EM algorithm* was originally presented by Dempster et al. in 1977 [13]. They presented it as a method of doing maximum likelihood estimation for incomplete data. Their work formalised and generalised several old ideas. For example Baum et al. [3] used very similar methods with hidden Markov models several years earlier.

The term "incomplete data" refers to a case where the complete data consists of two parts, only one of which is observed. Missing part can only be inferred based on how it affects the observed part. A typical example would be the generative model in Section 3.1.1, where the $\mathbf{x}(t)$ are the observed data $\boldsymbol{X}$ and the sources $\mathbf{s}(t)$ are the missing data $\boldsymbol{S}$.

The same algorithm can be easily interpreted in the Bayesian framework. Let us assume that we are trying to estimate the posterior probability $p(\boldsymbol{\theta}|\boldsymbol{X})$ for some model parameters $\boldsymbol{\theta}$. In such a case, the Bayesian version gives a point estimate for posterior maximum of $\boldsymbol{\theta}$.

The algorithm starts at some initial estimate $\widehat{\boldsymbol{\theta}}_0$. The estimate is improved iteratively by first computing the conditional probability distribution $p(\boldsymbol{S}|\widehat{\boldsymbol{\theta}}_i, \boldsymbol{X})$ of the missing data given the current estimate of the parameters. After this, a new estimate $\widehat{\boldsymbol{\theta}}_{i+1}$ for the parameters is computed by maximising the expectation of $\log p(\boldsymbol{\theta}|\boldsymbol{S}, \boldsymbol{X})$ over the previously calculated $p(\boldsymbol{S}|\widehat{\boldsymbol{\theta}}_i, \boldsymbol{X})$. The first step is called the expectation step (E-step) and the latter is called maximisation step (M-step) [16, 57].

An important feature in the EM algorithm is that it gives the complete probability distribution for the missing data and uses point estimates only for the model parameters. It should therefore give better results than simple MAP estimation.

Neal and Hinton showed in [45] that the EM algorithm can be interpreted as minimisation of the cost function given by

$$C = E_{q(\boldsymbol{S})}\left[\log \frac{q(\boldsymbol{S})}{p(\boldsymbol{X}, \boldsymbol{S}|\boldsymbol{\theta})}\right] = E_{q(\boldsymbol{S})}\left[\log \frac{q(\boldsymbol{S})}{p(\boldsymbol{S}|\boldsymbol{X}, \boldsymbol{\theta})}\right] - \log p(\boldsymbol{X}|\boldsymbol{\theta}) \qquad (3.10)$$

where at step t, $q(\boldsymbol{S}) = p(\boldsymbol{S}|\widehat{\boldsymbol{\theta}}_{t-1}, \boldsymbol{X})$. The notation $E_{q(\boldsymbol{S})}[\cdots]$ means that the expectation is taken over the distribution $q(\boldsymbol{S})$. This interpretation can be used to justify many interesting variants of the algorithm.

3.3 Ensemble learning

Ensemble learning is a relatively new concept suggested by Hinton and van Camp in 1993 [23]. It allows approximating the true posterior distribution with a tractable approximation and fitting it to the actual probability mass with no intermediate point estimates.

The posterior distribution of the model parameters $\boldsymbol{\theta}$, $p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H})$, is approximated with another distribution or *approximating ensemble* $q(\boldsymbol{\theta})$. The objective function chosen to measure the quality of the approximation is essentially the same cost function as the one for EM algorithm in Equation (3.10) [38]

$$C(\boldsymbol{\theta}) = \mathrm{E}_{q(\boldsymbol{\theta})}\left\{\log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \boldsymbol{X}|\mathcal{H})}\right\} = \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \boldsymbol{X}|\mathcal{H})} d\boldsymbol{\theta}. \qquad (3.11)$$

Ensemble learning is based on finding an optimal function to approximate another function. Such optimisation methods are called *variational methods* and therefore ensemble learning is sometimes also called *variational learning* [30].

A closer look at the cost function $C(\boldsymbol{\theta})$ shows that it can be represented as a sum of two simple terms

$$\begin{aligned} C(\boldsymbol{\theta}) &= \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \boldsymbol{X}|\mathcal{H})} d\boldsymbol{\theta} = \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H}) p(\boldsymbol{X}|\mathcal{H})} d\boldsymbol{\theta} \\ &= \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H})} d\boldsymbol{\theta} - \log p(\boldsymbol{X}|\mathcal{H}). \end{aligned} \qquad (3.12)$$

The first term in Equation (3.12) is the *Kullback–Leibler divergence* between the approximate posterior $q(\boldsymbol{\theta})$ and the true posterior $p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H})$. A simple application of Jensen's inequality [52] shows that the Kullback–Leibler divergence $D(q||p)$ between two distributions $q(\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$ is always nonnegative:

$$\begin{aligned} -D(q(\boldsymbol{\theta})||p(\boldsymbol{\theta})) &= \int q(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} \\ &\leq \log \int q(\boldsymbol{\theta}) \frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} = \log \int p(\boldsymbol{\theta}) d\boldsymbol{\theta} = \log 1 = 0. \end{aligned} \tag{3.13}$$

Since the logarithm is a strictly concave function, the equality holds if and only if $p(\boldsymbol{\theta})/q(\boldsymbol{\theta}) = const.$, i.e. $p(\boldsymbol{\theta}) = q(\boldsymbol{\theta})$.

The Kullback–Leibler divergence is not symmetric and it does not obey the triangle inequality, so it is not a metric. Nevertheless it can be considered a kind of a distance measure between probability distributions [12].

Using the inequality in Equation (3.13) we find that the cost function $C(\boldsymbol{\theta})$ is bounded from below by the negative logarithm of the evidence

$$C(\boldsymbol{\theta}) = D(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H})) - \log p(\boldsymbol{X}|\mathcal{H}) \geq -\log p(\boldsymbol{X}|\mathcal{H}) \tag{3.14}$$

and there is equality if and only if $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H})$.

Looking at this the other way round, the cost function gives a lower bound on the model evidence with

$$p(\boldsymbol{X}|\mathcal{H}) \geq e^{-C(\boldsymbol{\theta})}. \tag{3.15}$$

The error of this estimate is governed by $D(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\boldsymbol{X}, \mathcal{H}))$. Assuming the distribution $q(\boldsymbol{\theta})$ has been optimised to fit well to the true posterior, the error should be rather small. Therefore it is possible to approximate the evidence by $p(\boldsymbol{X}|\mathcal{H}) \approx e^{-C(\boldsymbol{\theta})}$. This allows using the values of the cost function for model selection as presented in Section 3.2.1 [35].

An important feature for practical use of ensemble learning is that the cost function and its derivatives with respect to the parameters of the approximating distribution can be easily evaluated for many models. Hinton and van Camp [23] used a separable Gaussian approximating distribution for a single hidden layer MLP network. After that many authors have used the method for different applications.

3.3.1 Information theoretic approach

In their original paper [23], Hinton and van Camp approached ensemble learning from an information theoretic point of view by using the *Minimum Description Length (MDL) Principle* [61]. They developed a new coding method for noisy parameter values which led to the cost function of Equation (3.11). This allows interpreting the cost $C(\boldsymbol{\theta})$ in Equation (3.11) as a description length for the data using the chosen model.

The MDL principle asserts that the best model for given data is the one that attains the shortest description of the data. The description length can be evaluated in bits and it represents the length of the message needed to transmit the data. The idea is that one builds a model for the data and then sends the description of that model and the residual of the data that could not be modelled. Thus the total description length is $L(\text{data}) = L(\text{model}) + L(\text{error})$.

The code length is related to probability because according to the coding theorem, an event x_1 having probability $p(x_1)$ can be coded using $-\log_2 p(x_1)$ bits, assuming both the sender and the receiver know the distribution p.

In their article Hinton and van Camp developed a method for encoding the parameters of the model in such a way, that the expected code length is the one given by Equation (3.11). Derivation of this result can be found in the original paper by Hinton and van Camp [23] or in the doctoral thesis [57] by Harri Valpola.

Chapter 4

Building blocks of the model

The nonlinear switching state-space model is a combination of two well known models, the hidden Markov model (HMM) and the nonlinear state-space model (NSSM). In this chapter, a brief review of previous work on these models and their combinations is presented. Section 4.1 deals with HMMs and Section 4.2 with NSSMs. Section 4.3 discusses some of the combinations of the two models.

The HMM and linear state-space model (SSM) are actually rather closely related. They can both be interpreted as linear Gaussian models [50]. The greatest difference between the models is that the SSM has continuous hidden states whereas the HMM has only discrete states. Roweis and Ghahramani have written a review [50] that nicely shows the common properties of the models. This thesis will nevertheless follow the more standard formulations, which tend to hide these connections.

4.1 Hidden Markov models

Hidden Markov models (HMMs) are latent variable models based on the Markov chains. HMMs are widely used especially in speech recognition and there is an extensive literature on them. The tutorial by Rabiner [48] is often considered a definitive guide to HMMs in speech recognition. Ghahramani [17] gives another introduction with some more recent extensions to the basic model.

Before going into details of HMMs, some of the basic properties of Markov chains are first reviewd briefly.

4.1.1 Markov chains

A *Markov chain* is a discrete stochastic process with discrete states and discrete transformations between them. At each time instant the system is in one of the N possible states, numbered from one to N. At regularly spaced discrete times, the system switches its state, possibly back to the same state. The initial state of the chain is denoted M_1 and the states after each time of change are $M_2, M_3, \ldots$. Standard first order Markov chain has the additional property that the probabilities of the future states depend only on the current state and not the ones before it [48]. Formally this means that

$$p(M_{t+1} = j | M_t = i, M_{t-1} = k, \ldots) = p(M_{t+1} = j | M_t = i) \quad \forall t, i, j, k. \tag{4.1}$$

This is called the *Markov property* of the chain.

Because of the Markov property, the complete probability distribution of the states of a Markov chain is defined by the initial distribution $\pi_i = p(M_1 = i)$ and the state transition probability matrix

$$a_{ij} = p(M_{t+1} = j | M_t = i), \quad 1 \leq i, j \leq N. \tag{4.2}$$

Let us denote $\boldsymbol{\pi} = (\pi_i)$ and $\mathbf{A} = (a_{ij})$. In the general case the transition probabilities could be time dependent, i.e. $a_{ij} = a_{ij}(t)$, but in this thesis only the time independent case is considered.

This allows the evaluation of the probability of a sequence of states $\boldsymbol{M} = (M_1, M_2, \ldots, M_T)$, given the model parameters $\boldsymbol{\theta} = (\mathbf{A}, \boldsymbol{\pi})$, as

$$p(\boldsymbol{M}|\boldsymbol{\theta}) = \pi_{M_1} \left[\prod_{t=1}^{T-1} a_{M_t M_{t+1}} \right]. \tag{4.3}$$

4.1.2 Hidden states

Hidden Markov model is basically a Markov chain whose internal state cannot be observed directly but only through some probabilistic function. That is, the internal state of the model only determines the probability distribution of the observed variables.

Let us denote the observations by $\boldsymbol{X} = (x(1), \ldots, x(T))$. For each state, the distribution $p(x(t)|M_t)$ is defined and independent of the time index t. The exact form of this conditional distribution depends on the application. In the simplest case there is only a finite number of different observation symbols. In this case the distribution can be characterised by the point probabilities

$$b_i(m) = p(x(t) = m|M_t = i), \quad \forall i, m. \tag{4.4}$$

Letting $\mathbf{B} = (b_i(m))$ and the parameters $\boldsymbol{\theta} = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$, the joint probability of an observation sequence and a state sequence can be evaluated by simple extension to Equation (4.3)

$$p(\boldsymbol{X}, \boldsymbol{M}|\boldsymbol{\theta}) = \pi_{M_1} \left[\prod_{t=1}^{T-1} a_{M_t M_{t+1}}\right] \left[\prod_{t=1}^{T} b_{M_t}(x(t))\right]. \tag{4.5}$$

The posterior probability of a state sequence can be derived from this as

$$p(\boldsymbol{M}|\boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{P(\boldsymbol{X}|\boldsymbol{\theta})} \pi_{M_1} \left[\prod_{t=1}^{T-1} a_{M_t M_{t+1}}\right] \left[\prod_{t=1}^{T} b_{M_t}(x(t))\right] \tag{4.6}$$

where $P(\boldsymbol{X}|\boldsymbol{\theta}) = \sum_{\boldsymbol{M}} P(\boldsymbol{X}, \boldsymbol{M}|\boldsymbol{\theta})$.

4.1.3 Continuous observations

An obvious extension to the basic HMM model is to allow continuous observation space instead of a finite number of discrete symbols. In this model the parameters $\mathbf{B}$ cannot be described as a simple matrix of point probabilities but rather as a complete pdf over the continuous observation space for each state. Therefore the values of $b_i(m)$ in Equation (4.4) must be replaced with a continuous probability distribution

$$b_i(x(t)) = p(x(t)|M_t = i), \quad \forall x(t), i. \tag{4.7}$$

This model is called *continuous density hidden Markov model* (CDHMM). The probability of an observation sequence evaluated in Equation (4.5) stays the same. The conditional distributions $b_i(x(t))$ can in principle be arbitrary but usually they are restricted to be finite mixtures of simple parametric distributions, like Gaussians [48].

4.1.4 Learning algorithms

Rabiner [48] lists three basic problems for HMMs:

1. Given the observation sequence $\boldsymbol{X}$ and the model $\boldsymbol{\theta}$, how can the probability $p(\boldsymbol{X}|\boldsymbol{\theta})$ be computed efficiently?
2. Given the observation sequence $\boldsymbol{X}$ and the model $\boldsymbol{\theta}$, how can the corresponding optimal state sequence $\boldsymbol{M}$ be estimated?
3. Given the observation sequence $\boldsymbol{X}$, how can the model parameters $\boldsymbol{\theta}$ be optimised to better describe the observations?

Problem 1 is not a learning problem but rather one needed in evaluating the model. Its difficulty comes from the fact that one needs to calculate the sum over all the possible state sequences. This can be diverted by calculating the probabilities recursively with respect to the length of the observation sequence. This operation forms one half of the *forward–backward* algorithm and it is very typical for HMM calculations. The algorithm uses an auxiliary variable $\alpha_i(t)$ which is defined as

$$\alpha_i(t) = p(x_{1:t}, M_t = i|\boldsymbol{\theta}). \tag{4.8}$$

Here we have used the notation $x_{1:t} = (x(1), \ldots, x(t))$.

The algorithm to evaluate $p(\boldsymbol{X}|\boldsymbol{\theta})$ proceeds as follows:

1. Initialisation:
$$\alpha_j(1) = b_j(x(1))\pi_j$$
2. Iteration:
$$\alpha_j(t+1) = \left[\sum_{i=1}^{N} \alpha_i(t) a_{ij}\right] b_j(x(t+1))$$
3. Termination:
$$p(\boldsymbol{X}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \alpha_i(T).$$

The solution of Problem 2 is much more difficult. Rabiner uses classical statistics and interprets it as a maximum likelihood estimation problem for the state sequence, the solution of which is given by the *Viterbi algorithm*. A Bayesian solution to the problem can be found with a simple modification of the solution of the next problem which essentially uses the posterior of the hidden states.

Rabiner's statement of Problem 3 is more precise than ours and asks for the maximum likelihood solution optimising $p(\boldsymbol{X}|\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. This problem is solved by the *Baum–Welch* algorithm which is an application of the EM algorithm to the problem.

The Baum–Welch algorithm uses the complete forward–backward procedure with a backward pass to compute $\beta_i(t) = p(x_{t:T}|M_t = i, \boldsymbol{\theta})$. This can be done very much like the forward pass:

1. Initialisation:
$$\beta_i(T) = b_i(x(T))$$
2. Iteration:
$$\beta_i(t) = b_i(x(t)) \left[\sum_{j=1}^{N} a_{ij}\beta_j(t+1) \right].$$

The M-step of Baum–Welch algorithm can be expressed in terms of $\eta_{ij}(t)$, the posterior probability that there was a transition between state i and state j at time step t given $\boldsymbol{X}$ and $\boldsymbol{\theta}$. This probability can be easily calculated with forward and backward variables α and β:

$$\eta_{ij}(t) = p(M_t = i, M_{t+1} = j|\boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{Z_n}\alpha_i(t)a_{ij}\beta_j(t+1) \tag{4.9}$$

where Z_n is a normalising constant such that $\sum_{i,j=1}^{N} \eta_{ij}(t) = 1$. Then the M-step for a_{ij} is

$$a_{ij} = \frac{1}{Z'_n}\sum_{t=1}^{T-1} \eta_{ij}(t) \tag{4.10}$$

where Z'_n is another constant needed to make the transition probabilities normalised. There are similar update formulas for other parameters. The algorithm can also easily be extended to take into account the possible priors of different variables [39].

Ensemble learning for hidden Markov models

MacKay showed in [39] how ensemble learning can be applied to learning of HMMs with discrete observations. With suitable priors for all the variables, the problem can be solved analytically and the resulting algorithm turns out to be a rather simple modification of the Baum–Welch algorithm.

MacKay uses Dirichlet distributions as priors for the model parameters $\boldsymbol{\theta}$. (See Appendix A for the definition of the Dirichlet distribution and some of its properties.) The likelihood of the model is discrete with respect to all the parameters and the Dirichlet distribution is the conjugate prior of such a distribution. Because of this, the posterior distributions of all the parameters are also Dirichlet. The update rule for the parameters W_{ij} of the Dirichlet distribution of the transition probabilities is

$$W_{ij} = \sum_{t=1}^{T-1} \sum_{\boldsymbol{M}} q(\boldsymbol{M}) \delta(M_t = i, M_{t+1} = j) + u_{a_{ij}} \tag{4.11}$$

where $u_{a_{ij}}$ are the parameters of the prior and $q(\boldsymbol{M})$ is the approximate posterior used in ensemble learning. The update rules for other parameters are similar.

The posterior distribution of the hidden state probabilities turns out to be of exactly the same form as the likelihood in Equation (4.6) but with a_{ij} replaced with $a^*_{ij} = \exp\left(\mathrm{E}_{q(\mathbf{A})}\{\ln a_{ij}\}\right)$ and similarly for $b_i(x(t))$ and π_i. The required expectations over the Dirichlet distribution can be evaluated as in Equation (A.13) in Appendix A.

4.2 Nonlinear state-space models

The fundamental goal of this work has been to extend the nonlinear state-space model (NSSM) and learning algorithm for it developed by Dr. Harri Valpola [58, 60], by adding the possibility of several distinct states for the dynamical system, as governed by the HMM. The NSSM algorithm is an extension to earlier nonlinear factor analysis (NFA) algorithm [34].

In this section, some of the previous work on these models is reviewed briefly. This complements the discussion in Section 2.2.2 as the problem to be solved is essentially the same. But first let us begin with an introduction to linear state-space models (SSMs), a generalisation of which the nonlinear version is.

4.2.1 Linear models

The linear state-space model (SSM) is built from a standard linear dynamical system in Equation (2.4). However, the state is not observed directly but only through another linear mapping. The basic model can be described by a pair of equations:

$$\begin{aligned} \mathbf{s}(t+1) &= \mathbf{B}\mathbf{s}(t) + \mathbf{m}(t) \\ \mathbf{x}(t) &= \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t) \end{aligned} \tag{4.12}$$

where $\mathbf{x}(t) \in \mathbb{R}^n,\ t = 1, \ldots, T$ are the observations and $\mathbf{s}(t) \in \mathbb{R}^m,\ t = 1, \ldots, T$ are the hidden internal states of the dynamical system. These internal states stay in the so called *state-space* — hence the name state-space model. Vectors $\mathbf{m}(t)$ and $\mathbf{n}(t)$ are the process and the observation noise, respectively. The mappings $\mathbf{A}$ and $\mathbf{B}$ are the linear observation and prediction mappings.

As we saw in Section 2.1.2, the possible dynamics of such a linear model are very restricted. The process noise may help in keeping the whole system from converging to a single point, but still the system cannot explain any interesting complex phenomena.

Linear SSMs are nevertheless used widely because they are very efficient to use and provide a good enough short-term approximation for many cases even if the true dynamics are nonlinear.

The most famous variant of linear SSMs is the *Kalman filter* [31, 41, 32]. It gives a learning algorithm for the model defined by Equation (4.12) assuming all the parameters are Gaussian and certain independence assumptions are met. Kalman filtering is used in several application because it is easy to implement and efficient.

4.2.2 Extension from linear to nonlinear

The nonlinear state-space model is in principle a very simple extension of the linear model. All that needs to be done is to replace the linear mappings $\mathbf{A}$ and $\mathbf{B}$ of Equation (4.12) with general nonlinear mappings to get

$$\begin{aligned} \mathbf{s}(t+1) &= \mathbf{g}(\mathbf{s}(t)) + \mathbf{m}(t) \\ \mathbf{x}(t) &= \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t), \end{aligned} \tag{4.13}$$

where $\mathbf{f}$ and $\mathbf{g}$ are sufficiently smooth nonlinear mappings and all the other variables are as in Equation (4.12).

One of the greatest difficulties with the nonlinear model is that while a linear mapping $\mathbf{A} : \mathbb{R}^m \to \mathbb{R}^n$ can be uniquely determined by $m \cdot n$ real numbers (the elements of the corresponding matrix), there is no way to do the same for nonlinear mappings. Representing an arbitrary nonlinear mapping with even moderate accuracy requires much more parameters.

While the matrix form provides a natural parameterisation for linear mappings, no such evident approach exists for nonlinear mappings. There are several ways to approximate a given nonlinear mapping through different kinds of series decompositions. Unfortunately such parameterisations are often best suited for some special classes of functions or are very sensitive to the data and thus useless in the presence of noise. For example in polynomial approximations, the coefficients become very sensitive to the data when the degree of the approximating polynomial is raised. Trigonometric expansions (Fourier series) are well suited only for modelling periodic functions.

In neural network literature there are two competing nonlinear function approximation methods that are both widely used: *radial–basis function* (RBF) and *multilayer perceptron* (MLP) networks. They are both universal function approximators, meaning that given enough "neurons" they can model any function to the desired degree of accuracy [21].

In this work we use only MLP networks, so they are now discussed in greater detail.

4.2.3 Multilayer perceptrons

An MLP is a network of simple *neurons* called *perceptrons*. The basic concept of a single perceptron was introduced by Rosenblatt in 1958. The perceptron computes a single *output* from multiple real-valued *inputs* by forming a linear combination according to its input *weights* and then possibly putting the output through some nonlinear activation function. Mathematically this can be written as

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(\mathbf{w}^T \mathbf{x} + b) \tag{4.14}$$

where $\mathbf{w}$ denotes the vector of weights, $\mathbf{x}$ is the vector of inputs, b is the bias and φ is the activation function. A signal-flow graph of this operation is shown in Figure 4.1 [21, 5].

The original Rosenblatt's perceptron used a Heaviside step function as the activation function φ. Nowadays, and especially in multilayer networks, the activation function is often chosen to be the logistic sigmoid $1/(1+e^{-x})$ or the hyperbolic tangent $\tanh(x)$. They are related by $(\tanh(x)+1)/2 = 1/(1+e^{-2x})$. These functions are used because they are mathematically convenient and are close to linear near origin while saturating rather quickly when getting away from the origin. This allows MLP networks to model well both strongly and mildly nonlinear mappings.

A single perceptron is not very useful because of its limited mapping ability. No matter what activation function is used, the perceptron is only able to represent an oriented ridge-like function. The perceptrons can, however, be used as building blocks of a larger, much more practical structure. A typical

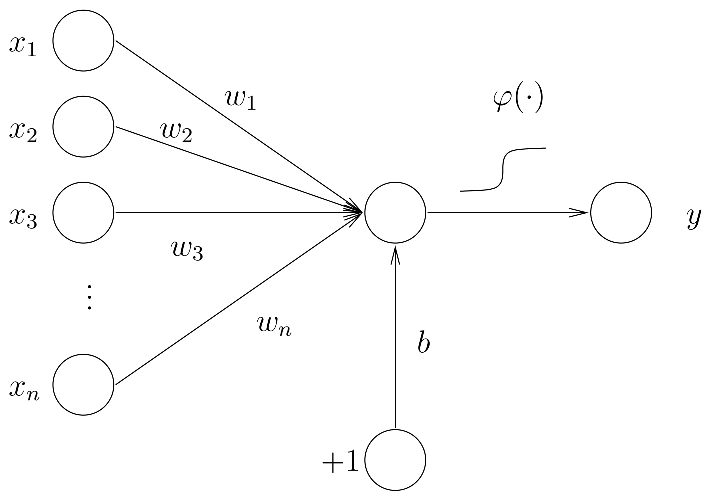


Figure 4.1: Signal-flow graph of the perceptron

multilayer perceptron (MLP) network consists of a set of source nodes forming the *input layer*, one or more *hidden layers* of computation nodes, and an *output layer* of nodes. The input signal propagates through the network layer-by-layer. The signal-flow of such a network with one hidden layer is shown in Figure 4.2 [21].

The computations performed by such a feedforward network with a single hidden layer with nonlinear activation functions and a linear output layer can be written mathematically as

$$\mathbf{x} = \mathbf{f}(\mathbf{s}) = \mathbf{B}\boldsymbol{\varphi}(\mathbf{A}\mathbf{s} + \mathbf{a}) + \mathbf{b} \tag{4.15}$$

where $\mathbf{s}$ is a vector of inputs and $\mathbf{x}$ a vector of outputs. $\mathbf{A}$ is the matrix of weights of the first layer, $\mathbf{a}$ is the bias vector of the first layer. $\mathbf{B}$ and $\mathbf{b}$ are, respectively, the weight matrix and the bias vector of the second layer. The function $\boldsymbol{\varphi}$ denotes an elementwise nonlinearity. The generalisation of the model to more hidden layers is obvious.

While single-layer networks composed of parallel perceptrons are rather limited in what kind of mappings they can represent, the power of an MLP network with only one hidden layer is surprisingly large. As Hornik et al. and Funahashi showed in 1989 [26, 15], such networks, like the one in Equation (4.15), are capable of approximating any continuous function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ to any given accuracy, provided that sufficiently many hidden units are available.

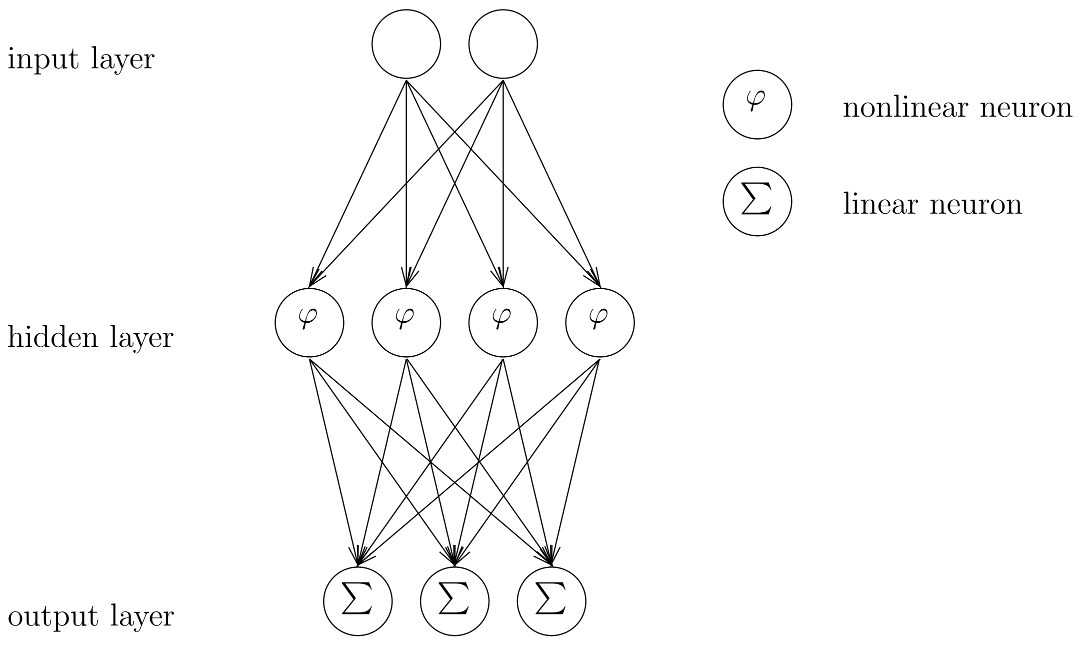


Figure 4.2: Signal-flow graph of an MLP

MLP networks are typically used in *supervised learning* problems. This means that there is a training set of input–output pairs and the network must learn to model the dependency between them. The training here means adapting all the weights and biases ($\mathbf{A}, \mathbf{B}, \mathbf{a}$ and $\mathbf{b}$ in Equation (4.15)) to their optimal values for the given pairs $(\mathbf{s}(t), \mathbf{x}(t))$. The criterion to be optimised is typically the squared reconstruction error $\sum_t ||\mathbf{f}(\mathbf{s}(t)) - \mathbf{x}(t)||^2$.

The supervised learning problem of the MLP can be solved with the *back-propagation algorithm*. The algorithm consists of two steps. In the *forward pass*, the predicted outputs corresponding to the given inputs are evaluated as in Equation (4.15). In the *backward pass*, partial derivatives of the cost function with respect to the different parameters are propagated back through the network. The chain rule of differentiation gives very similar computational rules for the backward pass as the ones in the forward pass. The network weights can then be adapted using any gradient-based optimisation algorithm. The whole process is iterated until the weights have converged [21].

The MLP network can also be used for unsupervised learning by using the so called *auto-associative* structure. This is done by setting the same values for both the inputs and the outputs of the network. The extracted sources emerge from the values of the hidden neurons [24]. This approach is computationally rather intensive. The MLP network has to have at least three hidden layers for any reasonable representation and training such a network is a time consuming process.

4.2.4 Nonlinear factor analysis

The NSSM implementation in [58] uses MLP networks to model the two nonlinear mappings in Equation (4.13). The learning procedure for the mappings is essentially the same as in the simpler NFA model [34], so it is presented first. The NFA model is also used in some of the experiments to explore the properties of the data set used.

Like the NSSM is a generalisation of the linear SSM, the NFA is a generalisation of the well-known linear factor analysis. The NFA can be defined with a generative data model given by

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t), \quad \mathbf{s}(t) = \mathbf{m}(t) \tag{4.16}$$

where the noise terms $\mathbf{n}(t)$ and $\mathbf{m}(t)$ are assumed to be Gaussian. The explicit mention of the factors being modeled as white noise is included to emphasise the difference between the NFA and the NSSM. The Gaussianity assumptions are made because of mathematical convenience, even though the Gaussianity of the factors is a serious limitation.

In the corresponding linear model the posterior will be an uncorrelated Gaussian, and in the nonlinear model it is approximated with a similar Gaussian. However, an uncorrelated Gaussian with equal variances in each direction is spherically symmetric and thus invariant with respect to all possible rotations of the factors. Even if the variances are not equal, there is a similar rotation invariance.

In traditional approaches to linear factor analysis this invariance is resolved by fixing the rotation using different heuristic criteria in choosing the optimal one. In the neural computation field the research has lead to algorithms like *independent component analysis* (ICA) [27] which uses basically the same linear generative model but with non-Gaussian factors. Similar techniques can also be applied to nonlinear model as discussed in [34].

The NFA algorithm uses an MLP network as the model of the nonlinearity $\mathbf{f}$. The model is learnt using ensemble learning. Most of the expectations needed for ensemble learning for such a model can be evaluated analytically. Only the terms involving the nonlinearity $\mathbf{f}$ must be approximated by using Taylor series approximation for the function about the posterior mean of the input.

The weights of the MLP network are updated with a back-propagation-like algorithm using the ensemble learning cost function. The unknown inputs of the network are also updated similarly, contrary to the standard supervised back-propagation.

All the parameters of the model are assumed to be Gaussian with hierarchical priors for most of them. The technical details of the model and the learning algorithm are covered in Chapters 5 and 6. Those chapters do, however, deal with the more general NSSM but the NFA emerges as a special case when all the temporal structure is ignored.

4.2.5 Learning algorithms

The most important NSSM learning algorithm for this work is the one by Dr. Harri Valpola [58, 60]. It uses MLP networks to model the nonlinearities and ensemble learning to optimise the model. It is discussed in detail in Sections 5.2 and 6.2 and it will therefore be skipped for now.

Even though it is not exactly a learning algorithm for complete NSSMs, the *extended Kalman filter* (EKF) is an important building block for many such algorithms. The EKF extends standard Kalman filtering for nonlinear models. The nonlinear functions $\mathbf{f}$ and $\mathbf{g}$ of Equation (4.13) must be known in advance. The algorithm works by linearising the functions about the estimated posterior mean. The posterior probability of the state variables is evaluated with a forward-backward type iteration by assuming the posterior of each sample to be Gaussian [42, 28, 32].

Briegel and Tresp [7] present an NSSM that uses MLP networks as the model of the nonlinearities. The learning algorithm is based on Monte-Carlo generalised EM algorithm, i.e. an EM algorithm with stochastic estimates for the conditional posteriors at different steps. The Monte-Carlo E-step is further optimised by generating the samples of the hidden states from an approximate Gaussian distribution instead of the true posterior. The approximate posterior is found using either the EKF or an alternative similar method.

Roweis and Ghahramani [51, 19] use RBF networks to model the nonlinearities. They use standard EM algorithm with EKF for the approximate E-step. The parameterisation of the RBF network allows an exact M-step for some of the parameters. All the parameters of the network can not, however, be adapted this way.

4.3 Previous hybrid models

There are numerous different hybrid models combining the HMM and a continuous model. As HMMs have been used most extensively in speech recognition, most of the hybrids have also been created for that purpose. Trentin and Gori [56] present a survey of speech recognition systems combining the HMM with some neural network model. These models are motivated by the end result of recognising speech by finding the correct HMM state sequence for the sequence of uttered phonemes. This is certainly one worthy goal but a good switching SSM can also be very useful in other time series modelling tasks.

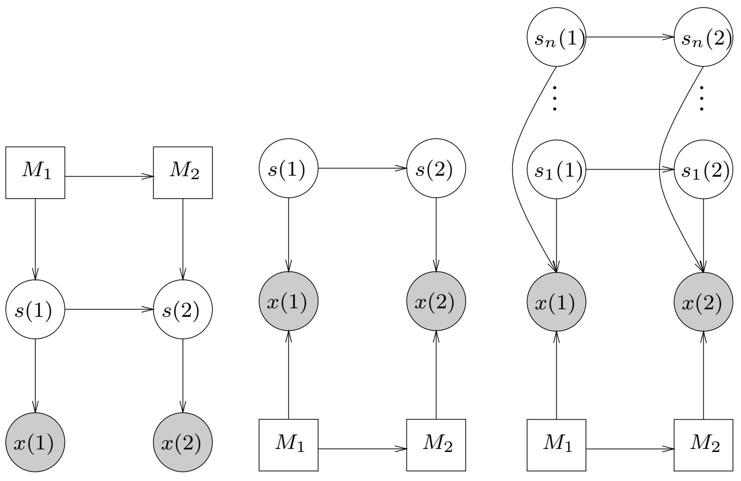
The basic idea of all the hybrids is to use the SSM or another continuous dynamical model to describe the short-term dynamics of the data while the HMM describes longer-term changes. In speech modelling, for instance, the HMM would govern the desired sequence of phonemes while the SSM models the actual production of the sound, the dynamics of the mouth, the vocal cord etc.

The models presented here can be divided into two classes. First there are the true switching SSMs, i.e. combinations of the HMM and a linear SSM. The second class consists of models that combine the HMM with another dynamical model which is a continuous one but not a true SSM.

4.3.1 Switching state-space models

There are several possible architectures for switching SSMs. Figure 4.3 shows some of the most basic ones [43]. The first subfigure corresponds to the case where the function $\mathbf{g}$ and possibly the model for noise $\mathbf{m}(t)$ in Equation (4.13) are different for different states. In the second subfigure, the function $\mathbf{f}$ and the noise $\mathbf{n}(t)$ depend on the switching variable. Some combination of these two approaches is of course also possible. The third subfigure shows an interesting architecture proposed by Ghahramani and Hinton [18] in which there are several completely separate SSMs and the switching variable chooses between them. Their model is especially interesting as it uses ensemble learning to infer the model parameters.

One of the problems with switching SSMs is that the exact E-step of the EM algorithm is intractable, even if the individual continuous hidden states are Gaussian. Assuming the HMM has N states, the posterior of a single state variable $\mathbf{s}(1)$ will be a mixture of N Gaussians, one for each HMM state M_1. When this is propagated forward according to the dynamical model, the



(a) Switching dynamics

(b) Switching observations

(c) Independent dynamical models [18]

Figure 4.3: Bayesian network representations of some switching state-space model architectures. The round nodes represent Gaussian variables and the square nodes are discrete. The shaded nodes are observed while the white ones are hidden.

mixture grows exponentially as the number of possible HMM state sequences increases. Finally, when the full observation sequence of length T is taken into account, the posterior of each $\mathbf{s}(t)$ will be a mixture of N^T Gaussians.

Ensemble learning is a very useful method in developing a tractable algorithm for the problem, although there are other heuristic methods for the same purpose. The other methods typically use some greedy procedure in collapsing the distribution and this may cause inaccuracies. This is not a problem with ensemble learning — it considers the whole sequence and minimises the Kullback–Leibler divergence, which in this case has no local minima.

4.3.2 Other hidden Markov model hybrids

To complete the review, two models that combine the HMM with another dynamical model that is not an SSM are now presented.

The hidden Markov ICA algorithm by Penny et al. [47] is an interesting variant of the switching SSMs. It has both switching dynamics and observation mapping, but the dynamics are not modelled with a linear dynamical system. Instead, the observations are transformed to *independent components* which are then each predicted separately with a generalised autoregressive (GAR) model. This allows using samples from further back for the prediction but everything is done strictly separately for all the components.

The MLP/HMM hybrid by Chung and Un [11] uses an MLP network for nonlinear prediction and an HMM to model its prediction errors. The predictor operates in the observation space so the model is not an NSSM. The prediction errors are modeled with a mixture-of-Gaussians for each HMM state. The model is trained using maximum likelihood and a discriminative criterion for minimal classification error in a speech recognition application.

Chapter 5

The model

In this chapter, it is shown how the two separate models of the previous chapter, the hidden Markov model (HMM) and the nonlinear state-space model (NSSM), can be combined into one switching NSSM. The chapter begins with Bayesian versions of the two individual models, the HMM in Section 5.1 and the NSSM in Section 5.2. The switching model is introduced in Section 5.3.

5.1 Bayesian continuous density hidden Markov model

In Section 4.1.4, a Bayesian formulation of the HMM based on the work of MacKay [39] is presented. This model is now extended for continuous observations instead of discrete ones.

Normally continuous density hidden Markov models (CDHMMs) use a mixture model like mixture-of-Gaussians as the distribution of the observations for a given state. This is, however, unnecessarily complicated for the HMM/NSSM hybrid so single Gaussians will be used as the observation model.

5.1.1 The model

The basic model is the same as the one presented in Section 4.1. The hidden state sequence is denoted by $\boldsymbol{M} = (M_1, \ldots, M_T)$ and other parameters by $\boldsymbol{\theta}$. The exact form of $\boldsymbol{\theta}$ will be specified later. The observations

$\boldsymbol{X} = (\mathbf{x}(1), \ldots, \mathbf{x}(T))$, given the corresponding hidden state, are assumed to be Gaussian with diagonal covariance matrix.

Given the HMM state sequence $\boldsymbol{M}$, the individual observations are assumed to be independent. Therefore the likelihood of the data can be written as

$$P(\boldsymbol{X}|\boldsymbol{M}, \boldsymbol{\theta}) = \prod_{t=1}^{T} \prod_{k=1}^{N} p(x_k(t)|M_t). \tag{5.1}$$

Because of the Markov property, the prior distribution of the probabilities of the hidden states can also be written in factorial form:

$$P(\boldsymbol{M}|\boldsymbol{\theta}) = p(M_1|\boldsymbol{\theta}) \prod_{t=1}^{T-1} p(M_{t+1}|M_t, \boldsymbol{\theta}). \tag{5.2}$$

The factors of Equations (5.1) and (5.2) are defined to be

$$\begin{aligned} p(x_k(t)|M_t = i) &= N(x_k(t);\ m_k(i), \exp(2v_k(i))) & (5.3)\\ p(M_{t+1} = j|M_t = i, \boldsymbol{\theta}) &= a_{ij} & (5.4)\\ p(M_1 = i|\boldsymbol{\theta}) &= \pi_i. & (5.5) \end{aligned}$$

The priors of all the parameters defined above are

$$\begin{aligned} p(\mathbf{a}_{i,\cdot}) &= \text{Dirichlet}(\mathbf{a}_{i,\cdot};\ \mathbf{u}_i^{(A)}) & (5.6)\\ p(\boldsymbol{\pi}) &= \text{Dirichlet}(\boldsymbol{\pi};\ \mathbf{u}^{(\pi)}) & (5.7)\\ p(m_k(i)) &= N(m_k(i);\ m_{m_k}, \exp(2v_{m_k})) & (5.8)\\ p(v_k(i)) &= N(v_k(i);\ m_{v_k}, \exp(2v_{v_k})). & (5.9) \end{aligned}$$

These should be written as conditional distributions conditional to the parameters of the hyperprior but the conditioning variables have been dropped out to simplify the notation.

The parameters $\mathbf{u}^{(\pi)}$ and $\mathbf{u}_i^{(A)}$ of the Dirichlet priors are fixed. Their values should be chosen to reflect true prior knowledge on the possible initial states and transition probabilities of the chain. In our example of speech recognition where the states of the HMM represent different phonemes, these values could, for instance, be estimated from textual data.

All the other parameters $m_{m_k}, v_{m_k}, m_{v_k}$ and v_{v_k} have higher hierarchical priors. As the number of parameters in such priors grows, only the full structure of the hierarchical prior of m_{m_k} is given. It is:

$$p(m_{m_k}) = N(m_{m_k};\ m_{m_m}, \exp(2v_{m_m})) \tag{5.10}$$
$$p(m_{m_m}) = N(m_{m_m};\ 0, 100^2) \tag{5.11}$$
$$p(v_{m_m}) = N(v_{m_m};\ 0, 100^2). \tag{5.12}$$

The hierarchical prior of for example $\mathbf{m}(i)$ can be summarised as follows:

- The different components of $\mathbf{m}(i)$ have different priors whereas the vectors corresponding to different states of the HMM share a common prior, which is parameterised with m_{m_n}.
- The parameters m_{m_n} corresponding to different components of the original vector $\mathbf{m}(i)$ share a common prior parameterised with m_{m_m}.
- The parameter m_{m_m} has a fixed noninformative prior.

The set of model parameters $\boldsymbol{\theta}$ consists of all these parameters and all the parameters of the hierarchical priors.

In the hierarchical structure formulated above, the Gaussian prior for the mean m of a Gaussian is a conjugate prior. Thus the posterior will also be Gaussian.

The parameterisation of the variance with $\sigma^2 = \exp(2v)$, $v \sim N(\alpha, \beta)$ is somewhat less conventional. The conjugate prior for variance of a Gaussian is the *inverse gamma* distribution. Adding a new level of hierarchy for the parameters of such a distribution would, however, be significantly more difficult. The present parameterisation allows adding similar layers of hierarchy for the parameters of the priors of m and v. In this parameterisation the posterior of v is not exactly Gaussian but it may be approximated with one. The exponential function will ensure that the variance will always be positive and the posterior will thus be closer to a Gaussian.

5.1.2 The approximating posterior distribution

The approximating posterior distribution needed in ensemble learning is over all the possible hidden state sequences $\boldsymbol{M}$ and the parameter values $\boldsymbol{\theta}$. The approximation is chosen to be of a factorial form

$$q(\boldsymbol{M}, \boldsymbol{\theta}) = q(\boldsymbol{M})q(\boldsymbol{\theta}). \tag{5.13}$$

The approximation $q(\boldsymbol{M})$ is a discrete distribution and it factorises as

$$q(\boldsymbol{M}) = q(M_1) \prod_{t=1}^{T-1} q(M_{t+1}|M_t). \tag{5.14}$$

The parameters of this distribution are the discrete probabilities $q(M_1 = i)$ and $q(M_{t+1} = i|M_t = j)$.

The distribution $q(\boldsymbol{\theta})$ is also formed as a product of independent distribution for different parameters. The parameters with Dirichlet priors have posterior approximations of a single Dirichlet distribution like for $\boldsymbol{\pi}$

$$q(\boldsymbol{\pi}) = \text{Dirichlet}(\boldsymbol{\pi};\ \hat{\boldsymbol{\pi}}), \tag{5.15}$$

or a product of Dirichlet distributions as for $\mathbf{A}$

$$q(\mathbf{A}) = \prod_{i=1}^{M} \text{Dirichlet}(\mathbf{a}_i;\ \hat{\mathbf{a}}_i). \tag{5.16}$$

These will actually be the optimal choices among all possible distributions, assuming the factorisation $q(\boldsymbol{M}, \boldsymbol{\pi}, \mathbf{A}) = q(\boldsymbol{M})q(\boldsymbol{\pi})q(\mathbf{A})$.

The parameters with Gaussian priors have Gaussian posterior approximations of the form

$$q(\theta_i) = N(\theta_i;\ \overline{\theta_i}, \widetilde{\theta_i}). \tag{5.17}$$

All these parameters are assumed to be independent.

5.2 Bayesian nonlinear state-space model

In this section, the NSSM model by Dr. Harri Valpola [58, 60] is presented. The corresponding learning algorithm for the model will be discussed in Section 6.2.

5.2.1 The generative model

The Bayesian model follows the standard NSSM defined in Equation (4.13):

$$\begin{aligned} \mathbf{s}(t+1) &= \mathbf{g}(\mathbf{s}(t)) + \mathbf{m}(t) \\ \mathbf{x}(t) &= \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t). \end{aligned} \tag{5.18}$$

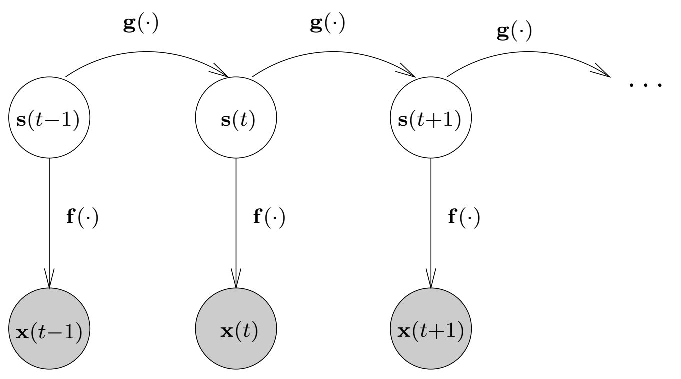


Figure 5.1: The nonlinear state-space model.

The basic structure of the model can be seen in Figure 5.1.

The nonlinear functions $\mathbf{f}$ and $\mathbf{g}$ of the NSSM are modelled by MLP networks. The networks have one hidden layer and can thus be written as in Equation (4.15):

$$\mathbf{f}(\mathbf{s}(t)) = \mathbf{B}\tanh[\mathbf{A}\mathbf{s}(t) + \mathbf{a}] + \mathbf{b} \tag{5.19}$$

$$\mathbf{g}(\mathbf{s}(t)) = \mathbf{s}(t) + \mathbf{D}\tanh[\mathbf{C}\mathbf{s}(t) + \mathbf{c}] + \mathbf{d} \tag{5.20}$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ are the weight matrices of the networks and $\mathbf{a}, \mathbf{b}, \mathbf{c}$ and $\mathbf{d}$ are the bias vectors.

Because the data are assumed to be generated by a continuous dynamical system, it is reasonable to assume that the values of the hidden states $\mathbf{s}(t)$ do not change very much from one time index to the next one. Therefore the MLP network representing the function $\mathbf{g}$ is used to model only the change.

5.2.2 The probabilistic model

Let us denote the observed data with $\boldsymbol{X} = (\mathbf{x}(1), \dots, \mathbf{x}(T))$, the hidden state values with $\boldsymbol{S} = (\mathbf{s}(1), \dots, \mathbf{s}(T))$ and all the other model parameters with $\boldsymbol{\theta}$. These other parameters consist of the weights and biases of the MLP networks and hyperparameters defining the prior distributions of other parameters.

The likelihood

Let us assume that the noise terms $\mathbf{m}(t)$ and $\mathbf{n}(t)$ in Equation (5.18) are all independent at different time instants and Gaussian with zero mean. The different components of the vector are, however, allowed to have different variances as governed by their hyperparameters. Following the developments of Section 3.1.1, the model implies a likelihood for the data

$$p(\mathbf{x}(t)|\mathbf{s}(t), \boldsymbol{\theta}) = N(\mathbf{x}(t);\ \mathbf{f}(\mathbf{s}(t)), \mathrm{diag}[\exp(2\mathbf{v}_n)]) \tag{5.21}$$

where $\mathrm{diag}[\exp(2\mathbf{v}_n)]$ denotes a diagonal matrix with the elements of the vector $\exp(2\mathbf{v}_n)$ on the diagonal. The vector $\mathbf{v}_n$ is a hyperparameter that defines the variances of different components of the noise.

As the values of the noise at different time instants are independent, the full data likelihood can be written as

$$p(\boldsymbol{X}|\boldsymbol{S}, \boldsymbol{\theta}) = \prod_{t=1}^{T} p(\mathbf{x}(t)|\mathbf{s}(t), \boldsymbol{\theta}) = \prod_{t=1}^{T}\prod_{k=1}^{n} p(x_k(t)|\mathbf{s}(t), \boldsymbol{\theta}) \tag{5.22}$$

where the individual factors are of the form

$$p(x_k(t)|\mathbf{s}(t), \boldsymbol{\theta}) = N(x_k(t);\ f_k(\mathbf{s}(t)), \exp(2v_{n_k})). \tag{5.23}$$

The prior of the states $\boldsymbol{S}$

Similarly, the prior of the states can be written as

$$\begin{aligned} p(\boldsymbol{S}|\boldsymbol{\theta}) &= p(\mathbf{s}(1)|\boldsymbol{\theta}) \prod_{t=1}^{T-1} p(\mathbf{s}(t+1)|\mathbf{s}(t), \boldsymbol{\theta}) \\ &= \prod_{k=1}^{m} p(s_k(1)|\boldsymbol{\theta}) \prod_{t=1}^{T-1} p(s_k(t+1)|\mathbf{s}(t), \boldsymbol{\theta}) \end{aligned} \tag{5.24}$$

where

$$p(s_k(t+1)|\mathbf{s}(t), \boldsymbol{\theta}) = N(s_k(t+1);\ g_k(\mathbf{s}(t)), \exp(2v_{m_k})) \tag{5.25}$$

and

$$p(s_k(1)|\boldsymbol{\theta}) = N(s_k(1);\ m_{s_k^0}, \exp(2v_{s_k^0})). \tag{5.26}$$

The prior of the parameters $\boldsymbol{\theta}$

Let us denote the elements of the weight matrices of the MLP networks by $\mathbf{A} = (A_{ij}), \mathbf{B} = (B_{ij}), \mathbf{C} = (C_{ij})$ and $\mathbf{D} = (D_{ij})$. The bias vectors consist similarly of elements $\mathbf{a} = (a_i), \mathbf{b} = (b_i), \mathbf{c} = (c_i)$ and $\mathbf{d} = (d_i)$.

All the elements of the weight matrices and the bias vectors are assumed to be independent and Gaussian. Their priors are as follows:

$$p(A_{ij}) = N(A_{ij};\ 0, 1) \tag{5.27}$$
$$p(B_{ij}) = N(B_{ij};\ 0, \exp(2v_{B_j})) \tag{5.28}$$
$$p(a_i) = N(a_i;\ m_a, \exp(2v_a)) \tag{5.29}$$
$$p(b_i) = N(b_i;\ m_b, \exp(2v_b)) \tag{5.30}$$
$$p(C_{ij}) = N(C_{ij};\ 0, \exp(2v_{C_i})) \tag{5.31}$$
$$p(D_{ij}) = N(D_{ij};\ 0, \exp(2v_{D_j})) \tag{5.32}$$
$$p(c_i) = N(c_i;\ m_c, \exp(2v_c)) \tag{5.33}$$
$$p(d_i) = N(d_i;\ m_d, \exp(2v_d)). \tag{5.34}$$

These distributions should again be written conditional to the corresponding hyperparameters, but the conditioning variables have been here omitted to keep the notation simpler.

Each of the bias vectors has a hierarchical prior that is shared among the different elements of that particular vector. The hyperparameters m_a, m_b, m_c, m_d, v_a, v_b, v_c and v_d all have zero mean Gaussian priors with standard deviation 100, which is a flat, essentially noninformative prior.

The structure of the priors of the weight matrices is much more interesting. The prior of $\mathbf{A}$ is chosen to be fixed to resolve a scaling indeterminacy between the hidden states $\mathbf{s}(t)$ and the weights of the MLP networks. This is evident from Equation (5.19) where any scaling in one of these parameters could be compensated by the other without affecting the results in any way. The other weight matrices $\mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ have zero mean priors with common variance for all the weights related to a single hidden neuron.

The remaining variance parameters from the priors of the weight matrices and from Equations (5.23), (5.25) and (5.26) again have hierarchical priors defined as

$$p(v_{B_j}) = N(v_{B_j};\ m_{v_B}, \exp(2v_{v_B})) \tag{5.35}$$
$$p(v_{C_i}) = N(v_{C_i};\ m_{v_C}, \exp(2v_{v_C})) \tag{5.36}$$
$$p(v_{D_j}) = N(v_{D_j};\ m_{v_D}, \exp(2v_{v_D})) \tag{5.37}$$
$$p(v_{n_k}) = N(v_{n_k};\ m_{v_n}, \exp(2v_{v_n})) \tag{5.38}$$
$$p(v_{m_k}) = N(v_{m_k};\ m_{v_m}, \exp(2v_{v_m})) \tag{5.39}$$
$$p(v_{s_k^0}) = N(v_{s_k^0};\ m_{v_s^0}, \exp(2v_{v_s^0})). \tag{5.40}$$

The prior distributions of the parameters of these distributions are again zero mean Gaussians with standard deviation 100.

5.2.3 The approximating posterior distribution

As with the HMM, the approximating posterior distribution is chosen to have a factorial form

$$q(\boldsymbol{S}, \boldsymbol{\theta}) = q(\boldsymbol{S})q(\boldsymbol{\theta}) = q(\boldsymbol{S}) \prod_i q(\theta_i). \tag{5.41}$$

The independent distributions for the parameters θ_i are all Gaussian with

$$q(\theta_i) = N(\theta_i;\ \overline{\theta}_i, \widetilde{\theta}_i) \tag{5.42}$$

where $\overline{\theta}_i$ and $\widetilde{\theta}_i$ are the variational parameters whose values must be optimised to minimise the cost function.

Because of the strong temporal correlations between the source values at consecutive time instants, the same approach cannot be applied to form $q(\boldsymbol{S})$. Therefore the approximation is chosen to be of the form

$$q(\boldsymbol{S}) = \prod_{k=1}^{n} \left[q(s_k(1)) \prod_{t=1}^{T-1} q(s_k(t+1)|s_k(t)) \right] \tag{5.43}$$

where the factors are again Gaussian.

The distributions for $s_k(1)$ can be handled as before with

$$q(s_k(1)) = N(s_k(1);\ \overline{s}_k(1), \widetilde{s}_k(1)). \tag{5.44}$$

The conditional distribution $q(s_k(t+1)|s_k(t))$ must be modified slightly to include the contribution of the previous state value. Saving the notation $\widetilde{s}_k(t)$ for the marginal variance of $q(s_k(t))$, the variance of the conditional distribution is denoted with $\mathring{s}_k(t+1)$. The mean of the distribution,

$$\mu_{si}(t+1) = \overline{s}_k(t+1) + \breve{s}_k(t, t+1)[s_k(t) - \overline{s}_k(t)], \tag{5.45}$$

depends linearly on the previous state value $s_k(t)$. This yields

$$q(s_k(t+1)|s_k(t)) = N(s_k(t+1);\ \mu_{si}(t+1), \mathring{s}_k(t+1)). \tag{5.46}$$

The variational parameters of the distribution are thus the mean $\overline{s}_k(t+1)$, the linear dependence $\breve{s}_k(t, t+1)$ and the variance $\mathring{s}_k(t+1)$. It should be noted that this dependence is only to the same component of the previous state value. The posterior dependence between the different components is neglected.

The marginal distribution of the states at time instant t may now be evaluated inductively starting from the beginning. Assuming $q(s_k(t-1)) = N(s_k(t-1);\ \overline{s}_k(t-1), \widetilde{s}_k(t-1))$, this yields

$$\begin{aligned} q(s_k(t)) &= \int q(s_k(t)|s_k(t-1))q(s_k(t-1))ds_k(t-1) \\ &= N(s_k(t);\ \overline{s}_k(t), \mathring{s}_k(t) + \breve{s}_k^2(t-1,t)\widetilde{s}_k(t-1)). \end{aligned} \tag{5.47}$$

Thus the marginal mean is the same as the conditional mean and the marginal variances can be computed using the recursion

$$\widetilde{s}_k(t) = \mathring{s}_k(t) + \breve{s}_k^2(t-1,t)\widetilde{s}_k(t-1). \tag{5.48}$$

5.3 Combining the two models

As we saw in Section 4.3, there are several possible ways to combine the HMM and the NSSM. The approach with switching dynamics seems more reasonable, so we shall concentrate on it. Hence the illustration of the NSSM in Figure 5.1 is augmented with the discrete HMM states to get Figure 5.2.

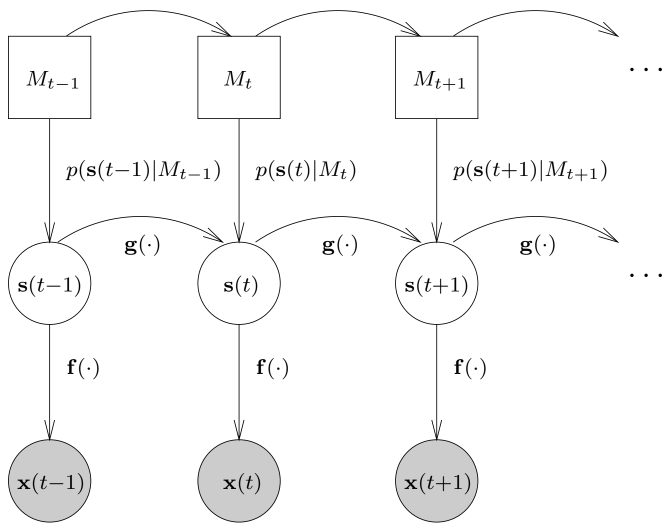


Figure 5.2: The switching nonlinear state-space model.

In the linear models, switching dynamics are often implemented by having a completely separate dynamical model for every HMM state. This is of course the most general approach. In the linear case such an approach is possible because of the simplicity of the individual linear models needed for each state. Unfortunately this is not so in the nonlinear case. Using more than just a few nonlinear models makes the system computationally far too heavy for any practical use.

To make things at least a little more computationally tractable, we use only one nonlinear mapping to describe the dynamics of all the states. Instead of own dynamics, every HMM state has its own characteristic model for the innovation process i.e. the description error of the dynamical mapping.

5.3.1 The structure of the model

The probabilistic interpretation of the dynamics of the sources in the standard NSSM is

$$p(\mathbf{s}(t)|\mathbf{s}(t-1), \boldsymbol{\theta}) = N(\mathbf{s}(t);\ \mathbf{g}(\mathbf{s}(t-1)), \mathrm{diag}[\exp(\mathbf{v}_m)]) \tag{5.49}$$

where $\mathbf{g}$ is the nonlinear dynamical mapping and $\mathrm{diag}[\exp(\mathbf{v}_m)]$ is the covariance matrix of the zero mean innovation process $\mathbf{i}(t) = \mathbf{s}(t) - \mathbf{g}(\mathbf{s}(t-1))$. The typical linear approach, applied to the nonlinear case, would use a different $\mathbf{g}$ and $\mathrm{diag}[\exp(\mathbf{v}_m)]$ for all the different states of the HMM.

The simplified model uses only one dynamical mapping $\mathbf{g}$ but has an own covariance matrix $\mathrm{diag}[\exp(\mathbf{v}_{M_j})]$ for each HMM state j. In addition to this, the innovation process is not assumed to be zero-mean but it has a mean depending on the HMM state. Mathematically this means that

$$p(\mathbf{s}(t)|\mathbf{s}(t-1), M_t = j, \boldsymbol{\theta}) = N(\mathbf{s}(t);\ \mathbf{g}(\mathbf{s}(t-1)) + \mathbf{m}_{M_j}, \mathrm{diag}[\exp(\mathbf{v}_{M_j})]) \tag{5.50}$$

where M_t is the HMM state, $\mathbf{m}_{M_j}$ and $\mathrm{diag}[\exp(\mathbf{v}_{M_j})]$ are, respectively, the mean and the covariance matrix of the innovation process for that state. The prior model of $s(1)$ remains unchanged.

Equation (5.50) summarises the differences between the switching NSSM and its components, as they were presented in Sections 5.1 and 5.2. The HMM "output" distribution is the one defined in Equation (5.50), not the data likelihood as in the "stand-alone" model. Similarly the model of the continuous hidden states in NSSM is slightly different from the one specified in Equation (5.25).

5.3.2 The approximating posterior distribution

The approximating posterior distribution is again chosen to be of a factorial form

$$q(\boldsymbol{M}, \boldsymbol{S}, \boldsymbol{\theta}) = q(\boldsymbol{M})q(\boldsymbol{S})q(\boldsymbol{\theta}). \tag{5.51}$$

The distributions of the state variables are as they were in the individual models, $q(\boldsymbol{M})$ as in Equation (5.14) and $q(\boldsymbol{S})$ as in Equations (5.43)–(5.46). The distribution of the parameters $\boldsymbol{\theta}$ is the product of corresponding distributions of the individual models, i.e. $q(\boldsymbol{\theta}) = q(\boldsymbol{\theta}_{\mathrm{HMM}})q(\boldsymbol{\theta}_{\mathrm{NSSM}})$.

There is one additional approximation in the choice of the form of q. This can be clearly seen from Equation (5.50). After marginalising over M_t, the conditional probability $p(\mathbf{s}(t)|\mathbf{s}(t{-}1), \boldsymbol{\theta})$ will be a mixture of as many Gaussians as there are states in the HMM. Marginalising out the past will result in an exponentially growing mixture thus making the problem intractable.

Our ensemble learning approach solves this problem by using only a single Gaussian as the posterior approximation $q(\mathbf{s}(t)|\mathbf{s}(t-1))$. This is of course just an approximation but it allows a tractable way to solve the problem.

Chapter 6

The algorithm

In this chapter, the learning algorithm used for optimising the parameters of the models defined in the previous chapter is presented. The structure of this chapter is essentially the same as in the previous chapter, i.e. the HMM is discussed in Section 6.1, the NSSM in Section 6.2 and finally the switching model in Section 6.3.

All the learning algorithms are based on ensemble learning which was introduced in Section 3.3. The sections of this chapter are mostly divided into two parts, one for evaluating the ensemble learning cost function and the other for optimising it.

6.1 Learning algorithm for the continuous density hidden Markov model

In this section, the ensemble learning cost function for the CDHMM model defined in Section 5.1 is derived and it is shown how its value can be optimised.

6.1.1 Evaluating the cost function

The general cost function of ensemble learning, as given in Equation (3.11), is

$$\begin{aligned} C(\boldsymbol{M},\boldsymbol{\theta}) = C_q + C_p &= \mathrm{E}\left[\log q(\boldsymbol{M},\boldsymbol{\theta})\right] + \mathrm{E}\left[-\log p(\boldsymbol{M},\boldsymbol{\theta},\boldsymbol{X})\right] \\ &= \mathrm{E}\left[\log q(\boldsymbol{M}) + \log q(\boldsymbol{\theta})\right] + \mathrm{E}\left[-\log p(\boldsymbol{X}|\boldsymbol{M},\boldsymbol{\theta})\right] \\ &\quad + \mathrm{E}\left[-\log p(\boldsymbol{M}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})\right] \end{aligned} \tag{6.1}$$

where all the expectations are taken over $q(\boldsymbol{M}, \boldsymbol{\theta})$. This will be true for all the similar formulas in this section unless explicitly stated otherwise.

The terms originating from the parameters $\boldsymbol{\theta}$

Assuming the parameters are $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_N\}$ and the approximation is of the form

$$q(\boldsymbol{\theta}) = \prod_{i=1}^{N} q(\theta_i), \tag{6.2}$$

the terms of Equation (6.1) originating from the parameters $\boldsymbol{\theta}$ can be written as

$$\mathrm{E}\left[\log q(\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})\right] = \sum_{i=1}^{N} \left(\mathrm{E}\left[\log q(\theta_i)\right] - \mathrm{E}\left[\log p(\theta_i)\right]\right). \tag{6.3}$$

In the case of Dirichlet distributions one θ_i in the previous equation must of course consist of a vector of parameters for the single distribution.

There are two different kinds of parameters in $\boldsymbol{\theta}$, those with Gaussian distribution and those with a Dirichlet distribution. In the Gaussian case the expectation $E[\log q(\theta_i)]$ over $q(\theta_i) = N(\theta_i;\ \overline{\theta_i}, \widetilde{\theta_i})$ gives the negative entropy of a Gaussian, $-1/2(1 + \log(2\pi\widetilde{\theta_i}))$, as derived in Equation (A.5) of Appendix A.

The expectation of $-\log p(\theta_i)$ can also be evaluated using the formulas of Appendix A. Assuming

$$p(\theta_i) = N(\theta_i;\ m, \exp(2v)) \tag{6.4}$$

where $q(\theta_i) = N(\theta_i;\ \overline{\theta_i}, \widetilde{\theta_i}), q(m) = N(m;\ \overline{m}, \widetilde{m})$ and $q(v) = N(v;\ \overline{v}, \widetilde{v})$, the expectation becomes

$$\begin{aligned} C_p(\theta_i) = \mathrm{E}\left[-\log p(\theta_i)\right] &= \mathrm{E}\left[\frac{1}{2}\log(2\pi\exp(2v)) + \frac{1}{2}(\theta_i - m)^2\exp(-2v)\right] \\ &= \frac{1}{2}\log(2\pi) + \mathrm{E}[v] + \frac{1}{2}\mathrm{E}\left[(\theta_i - m)^2\right]\mathrm{E}\left[\exp(-2v)\right] \\ &= \frac{1}{2}\log(2\pi) + \overline{v} + \frac{1}{2}\left[(\overline{\theta_i} - \overline{m})^2 + \widetilde{\theta_i} + \widetilde{m}\right]\exp(2\widetilde{v} - 2\overline{v}) \end{aligned} \tag{6.5}$$

where we have used the results of Equations (A.4) and (A.6).

For Dirichlet distributed parameters, the procedure is similar. Let us assume that the parameter $\mathbf{c} \in \boldsymbol{\theta}$, $p(\mathbf{c}) = \mathrm{Dirichlet}(\mathbf{c};\ \mathbf{u}^{(\mathbf{c})})$ and $q(\mathbf{c}) = \mathrm{Dirichlet}(\mathbf{c};\ \hat{\mathbf{c}})$.

Using the notation of Appendix A, the negative entropy of the Dirichlet distribution $q(\mathbf{c})$, $\mathrm{E}[\log q(\mathbf{c})]$, can be evaluated as in Equation (A.14) to yield

$$C_q(\mathbf{c}) = \mathrm{E}[\log q(\mathbf{c})] = \log Z(\hat{\mathbf{c}}) - \sum_{i=1}^{n} (\hat{c}_i - 1)[\Psi(\hat{c}_i) - \Psi(\hat{c}_0)]. \tag{6.6}$$

The special function required in these terms is $\Psi(x) = \frac{d}{dx}\ln(\Gamma(x))$, where $\Gamma(x)$ is the gamma function. The psi function $\Psi(x)$ is also known as the digamma function and it can be efficiently evaluated numerically for example using techniques described in [4]. The term $Z(\hat{\mathbf{c}})$ is a normalising constant of the Dirichlet distribution as defined in Appendix A.

The expectation of $-\log p(\mathbf{c})$ can be evaluated similarly

$$\begin{aligned} C_p(\mathbf{c}) &= -\mathrm{E}[\log p(\mathbf{c})] = -\mathrm{E}\left[\log\left(\frac{1}{Z(\mathbf{u}^{(\mathbf{c})})}\prod_{i=1}^{n} c_i^{u_i^{(\mathbf{c})}-1}\right)\right] \\ &= \log Z(\mathbf{u}^{(\mathbf{c})}) - \sum_{i=1}^{n}(u_i^{(\mathbf{c})} - 1)\,\mathrm{E}[\log c_i] \\ &= \log Z(\mathbf{u}^{(\mathbf{c})}) - \sum_{i=1}^{n}(u_i^{(\mathbf{c})} - 1)[\Psi(c_i) - \Psi(c_0)]. \end{aligned} \tag{6.7}$$

The likelihood term

The likelihood term in Equation (6.1) is rather easy to evaluate

$$\begin{aligned} C_p(X) &= -\mathrm{E}[\log p(\boldsymbol{X}|\boldsymbol{M}, \boldsymbol{\theta})] \\ &= \sum_{t=1}^{T}\sum_{i=1}^{M}\sum_{k=1}^{N} q(M_t = i)\,\mathrm{E}[-\log p(x_k(t)|M_t = i)] \\ &= \sum_{t=1}^{T}\sum_{i=1}^{M}\sum_{k=1}^{N} q(M_t = i)\bigg(\frac{1}{2}\log(2\pi) + \overline{v}_k(i) \\ &\quad + \frac{1}{2}\left[(x_k(t) - \overline{m}_k(i))^2 + \widetilde{m}_k(i)\right]\exp(2\widetilde{v}_k(i) - 2\overline{v}_k(i))\bigg) \end{aligned} \tag{6.8}$$

where we have used the result of Equation (6.5) for the expectation.

The terms originating from the hidden state sequence $\boldsymbol{M}$

The term $C_q(\boldsymbol{M})$ is just a sum over the discrete distribution. It can be further simplified into

$$\begin{aligned}
C_q(\boldsymbol{M}) &= \sum_{\boldsymbol{M}} q(\boldsymbol{M}) \log q(\boldsymbol{M}) = \sum_{\boldsymbol{M}} q(\boldsymbol{M}) \log \left(q(M_1) \prod_{t=1}^{T-1} q(M_{t+1}|M_t) \right) \\
&= \sum_{i=1}^{N} q(M_1 = i) \log q(M_1 = i) \\
&\quad + \sum_{t=1}^{T-1} \sum_{i,j=1}^{N} q(M_{t+1} = j, M_t = i) \log q(M_{t+1} = j | M_t = i).
\end{aligned} \tag{6.9}$$

The other term, $C_p(\boldsymbol{M})$ can be split down to

$$\begin{aligned}
C_p(\boldsymbol{M}) &= \mathrm{E}\left[-\log p(\boldsymbol{M}|\boldsymbol{\theta}) \right] = E\left[-\log p(M_1|\boldsymbol{\theta}) \right] + \sum_{t=1}^{T-1} E\left[-\log a_{M_t M_{t+1}} \right] \\
&= -\sum_{i=1}^{N} q(M_1 = i) E\left[\log \pi_i \right] - \sum_{t=1}^{T-1} \sum_{i,j=1}^{N} q(M_{t+1} = j, M_t = i) E\left[\log a_{ij} \right]
\end{aligned} \tag{6.10}$$

where according to Equation (A.13), $\mathrm{E}\left[\log \pi_i\right] = \Psi(\hat{\pi}_i) - \Psi(\sum_{j=1}^{N} \hat{\pi}_j)$ and similarly $\mathrm{E}\left[\log a_{ij}\right] = \Psi(\hat{a}_{ij}) - \Psi(\sum_{k=1}^{N} \hat{a}_{ik})$.

The above equations give the value of the cost function for given approximating distribution $q(\boldsymbol{\theta}, \boldsymbol{M})$. This value is important because it can be used to compare different models as shown in Section 3.3. Additionally it can be used to monitor whether the iterative optimisation procedure has converged.

6.1.2 Optimising the cost function

After defining the model and finding a way to evaluate the cost function, the next problem is to optimise the cost function. This will be done in a way that is very similar to the EM algorithm, i.e. by updating one part of the model at a time while keeping all the other parameters fixed. All the optimisation steps aim at finding a minimum of the cost function with the current values for fixed parameters. Since all the steps decrease the value of the cost function, the learning algorithm is guaranteed to converge.

Finding optimal $q(\boldsymbol{M})$

Assuming $q(\boldsymbol{\theta})$ is fixed, the cost function can be written, up to an additive constant, in the form

$$\begin{aligned}C(\boldsymbol{M}) &= \sum_{\boldsymbol{M}} q(\boldsymbol{M}) \Big[\log q(\boldsymbol{M}) - \int q(\boldsymbol{\theta}) \log p(\boldsymbol{M}|\boldsymbol{\theta}) d\boldsymbol{\theta} \\ &\qquad - \int q(\boldsymbol{\theta}) \log p(\boldsymbol{X}|\boldsymbol{M},\boldsymbol{\theta}) d\boldsymbol{\theta} \Big] \\ &= \sum_{\boldsymbol{M}} q(\boldsymbol{M}) \Big[\log q(\boldsymbol{M}) - \int q(\boldsymbol{\pi}) \log \pi_{M_1} d\boldsymbol{\pi} \\ &\qquad - \int \sum_{t=1}^{T-1} q(\mathbf{A}) \log a_{M_t M_{t+1}} d\mathbf{A} + \sum_{t=1}^{T} C(\mathbf{x}(t)|M_t) \Big]\end{aligned} \tag{6.11}$$

where $C(\mathbf{x}(t)|M_t)$ is the value of $\mathrm{E}[p(\mathbf{x}(t)|M_t, \boldsymbol{\theta})]$, i.e. the “cost” of current data sample given the HMM state.

By defining

$$\pi_i^* = \exp\left(\int q(\boldsymbol{\pi}) \log \pi_i d\boldsymbol{\pi} \right) \text{ and } a_{ij}^* = \exp\left(\int q(\mathbf{A}) \log a_{ij} d\mathbf{A} \right), \tag{6.12}$$

Equation (6.11) can be written in the form

$$C_{q(\boldsymbol{M})} = \sum_{\boldsymbol{M}} q(\boldsymbol{M}) \log \frac{q(\boldsymbol{M})}{\pi_{M_1}^* \left[\prod_{t=1}^{T-1} a_{M_t M_{t+1}}^* \right] \left[\prod_{t=1}^{T} \exp(-C(\mathbf{x}(t)|M_t)) \right]}. \tag{6.13}$$

The expression $\int q(x) \log \frac{q(x)}{p^*(x)}$ is minimised with respect to $q(x)$ by setting $q(x) = \frac{1}{Z} p^*(x)$ where Z is the appropriate normalising constant [39]. This can be proved with similar reasoning as in Equation (3.13).

The cost in Equation (6.13) can thus be minimised by setting

$$q(\boldsymbol{M}) = \frac{1}{Z_M} \pi_{M_1}^* \left[\prod_{t=1}^{T-1} a_{M_t M_{t+1}}^* \right] \left[\prod_{t=1}^{T} \exp(-C(\mathbf{x}(t)|M_t)) \right] \tag{6.14}$$

where Z_M is the appropriate normalising constant.

The derived optimal approximation is very similar in form to the exact posterior in Equation (4.6). Therefore the point probabilities of $q(M_1 = i)$ and

$q(M_t = j|M_{t-1} = i)$ can be evaluated with a modified forward–backward iteration. The result is the same as in Equation (4.9) except that in the iteration, π_i is replaced with π_i^*, a_{ij} is replaced with a_{ij}^* and $b_i(\mathbf{x})$ is replaced with $\exp(-C(\mathbf{x}(t)|M_t = i))$.

Finding optimal $q(\boldsymbol{\theta})$ for Dirichlet parameters

Let us now assume that $q(\boldsymbol{M})$ is fixed and optimise $q(\mathbf{A})$ and $q(\boldsymbol{\pi})$. Assuming that everything else is fixed, the cost function can be written as a functional of $q(\mathbf{A})$, up to an additive constant

$$\begin{aligned} C(\mathbf{A}) = \int q(\mathbf{A}) \Bigg[\log q(\mathbf{A}) - \sum_{i,j=1}^{N} (u_{ij}^{(A)} - 1) \log a_{ij} \\ - \sum_{\boldsymbol{M}} q(\boldsymbol{M}) \sum_{t=1}^{T-1} \log a_{M_t M_{t+1}} \Bigg] d\mathbf{A} \qquad (6.15) \\ = \int q(\mathbf{A}) \log \frac{q(\mathbf{A})}{\prod_{i,j=1}^{N} a_{ij}^{(W_{ij}-1)}} d\mathbf{A} \end{aligned}$$

where $W_{ij} = u_{ij}^{(A)} + \sum_{t=1}^{T-1} q(M_t = i, M_{t+1} = j)$.

As before, the optimal $q(\mathbf{A})$ is of the form $q(\mathbf{A}) = \frac{1}{Z_A} a_{ij}^{(W_{ij}-1)}$. The update rule for the parameters $\hat{a}_{ij}$ of $q(\mathbf{A})$ is thus

$$\hat{a}_{ij} \leftarrow W_{ij} = u_{ij}^{(A)} + \sum_{t=1}^{T-1} q(M_{t+1} = j|M_t = i). \qquad (6.16)$$

Similar reasoning for $\boldsymbol{\pi}$ gives the update rule

$$\hat{\pi}_i \leftarrow u_i^{(\pi)} + q(M_1 = i). \qquad (6.17)$$

Finding optimal $q(\boldsymbol{\theta})$ for Gaussian parameters

As an example of Gaussian parameters we shall consider $m_k(i)$ and $v_k(i)$. All the others are handled in essentially the same way except that there are no weights needed for different states.

To simplify the notation, all the indices from $m_k(i)$ and $v_k(i)$ are dropped out for the remainder of this section. The relevant terms of the cost function are now, up to an additive constant

$$\begin{aligned} C(m,v) = &\sum_{t=1}^{T} q(M_t = i)\left(\overline{v} + \frac{1}{2}\left[(x(t) - \overline{m})^2 + \widetilde{m}\right]\exp(2\widetilde{v} - 2\overline{v})\right) \\ &+ \frac{1}{2}\left[(\overline{m} - \overline{m}_m)^2 + \widetilde{m}\right]\exp(2\widetilde{v}_m - 2\overline{v}_m) - \frac{1}{2}\log\widetilde{m} \\ &+ \frac{1}{2}\left[(\overline{v} - \overline{m}_v)^2 + \widetilde{v}\right]\exp(2\widetilde{v}_v - 2\overline{v}_v) - \frac{1}{2}\log\widetilde{v}. \end{aligned} \tag{6.18}$$

Let us denote $\sigma^2_{\text{eff}} = \exp(2\overline{v} - 2\widetilde{v})$, $\sigma^2_{m,\text{eff}} = \exp(2\overline{v}_m - 2\widetilde{v}_m)$ and $\sigma^2_{v,\text{eff}} = \exp(2\overline{v}_v - 2\widetilde{v}_v)$.

The derivative of this expression with respect to $\widetilde{m}$ is easy to evaluate

$$\frac{\partial C}{\partial \widetilde{m}} = \sum_{t=1}^{T} q(M_t = i)\frac{1}{2\sigma^2_{\text{eff}}} + \frac{1}{2\sigma^2_{m,\text{eff}}} - \frac{1}{2\widetilde{m}}. \tag{6.19}$$

Setting this to zero gives

$$\widetilde{m} = \left(\sum_{t=1}^{T} q(M_t = i)\frac{1}{\sigma^2_{\text{eff}}} + \frac{1}{\sigma^2_{m,\text{eff}}}\right)^{-1}. \tag{6.20}$$

The derivative with respect to $\overline{m}$ is

$$\frac{\partial C}{\partial \overline{m}} = \sum_{t=1}^{T} q(M_t = i)\frac{1}{2\sigma^2_{\text{eff}}}\left[\overline{m} - x(t)\right] + \frac{1}{2\sigma^2_{m,\text{eff}}}\left[\overline{m} - \overline{m}_m\right] \tag{6.21}$$

which has a zero at

$$\overline{m} = \left[\sum_{t=1}^{T} q(M_t = i)\frac{1}{2\sigma^2_{\text{eff}}}x(t) + \frac{1}{2\sigma^2_{m,\text{eff}}}\overline{m}_m\right]\widetilde{m} \tag{6.22}$$

where $\widetilde{m}$ is given by Equation (6.20).

The solutions for parameters of $q(m)$ are exact. The true posterior for these parameters is also Gaussian so the approximation is equal to it. This is not the case for the parameters of $q(v)$. The true posterior for v is not Gaussian. The best Gaussian approximation with respect to the chosen criterion can still be found by solving the zero of the derivative of the cost function with respect to the parameters of $q(v)$. This is done using Newton's iteration.

The derivatives with respect to $\overline{v}$ and $\widetilde{v}$ are

$$\frac{\partial C}{\partial \overline{v}} = \sum_{t=1}^{T} q(M_t = i) \left(1 - \left[(x(t) - \overline{m})^2 + \widetilde{m}\right] \exp(2\widetilde{v} - 2\overline{v})\right) + \frac{\overline{v} - \overline{m}_v}{\sigma^2_{v,\mathrm{eff}}} \tag{6.23}$$

$$\frac{\partial C}{\partial \widetilde{v}} = \sum_{t=1}^{T} q(M_t = i) \left[(x(t) - \overline{m})^2 + \widetilde{m}\right] \exp(2\widetilde{v} - 2\overline{v}) + \frac{1}{2\sigma^2_{v,\mathrm{eff}}} + \frac{1}{2\widetilde{v}}. \tag{6.24}$$

These are set to zero and solved with Newton's iteration.

6.2 Learning algorithm for the nonlinear state-space model

In this section, the learning procedure for the NSSM defined in Section 5.2 is presented. The structure of the section is essentially the same as in the previous section on the HMM, i.e. it is first shown how to evaluate the cost function and then how to optimise it.

6.2.1 Evaluating the cost function

As before, the general cost function of ensemble learning, as given in Equation (3.11), is

$$\begin{aligned} C(\boldsymbol{S}, \boldsymbol{\theta}) &= C_q + C_p = \mathrm{E}\left[\log q(\boldsymbol{S}, \boldsymbol{\theta})\right] + \mathrm{E}\left[-\log p(\boldsymbol{S}, \boldsymbol{\theta}, \boldsymbol{X})\right] \\ &= \mathrm{E}\left[\log q(\boldsymbol{S}) + \log q(\boldsymbol{\theta})\right] + \mathrm{E}\left[-\log p(\boldsymbol{X}|\boldsymbol{S}, \boldsymbol{\theta})\right] \\ &\quad + \mathrm{E}\left[-\log p(\boldsymbol{S}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})\right] \end{aligned} \tag{6.25}$$

where the expectations are taken over $q(\boldsymbol{S}, \boldsymbol{\theta})$. This will be the case for the rest of the section unless stated otherwise.

In the NSSM, all the probability distributions involved are Gaussian so most of the terms will resemble the corresponding ones of the CDHMM. For the parameters $\boldsymbol{\theta}$,

$$C_q(\theta_i) = \mathrm{E}\left[\log q(\theta_i)\right] = -\frac{1}{2}(1 + \log(2\pi\widetilde{\theta}_i)). \tag{6.26}$$

The term $C_q(\boldsymbol{S})$ is a little more complicated:

$$C_q(\boldsymbol{S}) = \mathrm{E}\left[\log q(\boldsymbol{S})\right] = \sum_{k=1}^{n} \Big(\mathrm{E}\left[\log q(s_k(1))\right] + \sum_{t=1}^{T-1} \mathrm{E}\left[\log q(s_k(t+1)|s_k(t))\right] \Big). \tag{6.27}$$

The first term reduces to Equation (6.26) but the second term is a little different:

$$\begin{aligned} &E_{q(s_k(t),s_k(t+1))}\left[\log q(s_k(t+1)|s_k(t))\right] \\ &\qquad = E_{q(s_k(t))}\left\{E_{q(s_k(t+1)|s_k(t))}\left[\log q(s_k(t+1)|s_k(t))\right]\right\} \\ &= E_{q(s_k(t))}\left\{-\frac{1}{2}(1+\log(2\pi \mathring{s}_k(t+1)))\right\} = -\frac{1}{2}(1+\log(2\pi \mathring{s}_k(t+1))). \end{aligned} \tag{6.28}$$

The expectation of $-\log p(\theta_i|m,v)$ has been evaluated in Equation (6.5), so the only remaining terms are $\mathrm{E}\left[-\log p(\boldsymbol{X}|\boldsymbol{S},\boldsymbol{\theta})\right]$ and $\mathrm{E}\left[-\log p(\boldsymbol{S}|\boldsymbol{\theta})\right]$. They both involve the nonlinear mappings $\mathbf{f}$ and $\mathbf{g}$, so they cannot be evaluated exactly.

The formulas allowing to approximate the distribution of the outputs of an MLP network $\mathbf{f}$ are presented in Appendix B. As a result we get the posterior mean of the outputs $\overline{f}_k(\mathbf{s})$ and the posterior variance, decomposed as

$$\widetilde{f}_k(\mathbf{s}) \approx \widetilde{f}_k^*(\mathbf{s}) + \sum_j \widetilde{s}_j \left[\frac{\partial f_k(\mathbf{s})}{\partial s_j}\right]^2. \tag{6.29}$$

With these results the remaining terms of the cost function are relatively easy to evaluate. The likelihood term is a standard Gaussian and yields

$$\begin{aligned} C_p(x_k(t)) &= \mathrm{E}\left[-\log p(x_k(t)|\mathbf{s}(t),\boldsymbol{\theta})\right] \\ &= \frac{1}{2}\log(2\pi) + \overline{v}_{n_k} \\ &\quad + \frac{1}{2}\left[(x_k(t) - \overline{f}_k(\mathbf{s}(t)))^2 + \widetilde{f}_k(\mathbf{s}(t))\right]\exp(2\widetilde{v}_{n_k} - 2\overline{v}_{n_k}). \end{aligned} \tag{6.30}$$

The source term is more difficult. The problematic expectation is

$$
\begin{aligned}
\alpha_k(t) &= \mathrm{E}\left[(s_k(t) - g_k(\mathbf{s}(t-1)))^2\right] \\
&= \mathrm{E}\left[(\overline{s}_k(t) + \breve{s}_k(t-1,t)(s_k(t-1) - \overline{s}_k(t-1)) - g_k(\mathbf{s}(t-1)))^2\right] + \mathring{s}_k(t) \\
&= \overline{s}_k^2(t) + \mathring{s}_k(t) + \mathrm{E}\left[\breve{s}_k^2(t-1,t)(s_k(t-1) - \overline{s}_k(t-1))^2\right. \\
&\quad + g_k^2(\mathbf{s}(t-1)) + 2\overline{s}_k(t)\breve{s}_k(t-1,t)(s_k(t-1) - \overline{s}_k(t-1)) \\
&\quad \left. - 2\overline{s}_k(t) g_k(\mathbf{s}(t-1)) - 2\breve{s}_k(t-1,t)(s_k(t-1) - \overline{s}_k(t-1)) g_k(\mathbf{s}(t-1))\right] \\
&= \overline{s}_k^2(t) + \mathring{s}_k(t) + \breve{s}_k^2(t-1,t)\widetilde{s}_k(t-1) + \overline{g}_k^2(\mathbf{s}(t-1)) + \widetilde{g}_k(\mathbf{s}(t-1)) \\
&\quad - 2\overline{s}_k(t)\overline{g}_k(\mathbf{s}(t-1)) - 2\breve{s}_k(t-1,t)\frac{\partial g_k(\mathbf{s}(t-1))}{\partial s_k(t-1)}\widetilde{s}_k(t-1) \\
&= (\overline{s}_k(t) - \overline{g}(\mathbf{s}(t-1)))^2 + \widetilde{s}_k(t) + \widetilde{g}_k(\mathbf{s}(t-1)) \\
&\quad - 2\breve{s}_k(t-1,t)\frac{\partial g_k(\mathbf{s}(t-1))}{\partial s_k(t-1)}\widetilde{s}_k(t-1)
\end{aligned}
\tag{6.31}
$$

where we have used the additional approximation

$$
\begin{aligned}
&\mathrm{E}\left[\breve{s}_k(t-1,t)(s_k(t-1) - \overline{s}_k(t-1)) g_k(\mathbf{s}(t-1))\right] = \\
&\qquad\qquad \breve{s}_k(t-1,t)\frac{\partial g_k(\mathbf{s}(t-1))}{\partial s_k(t-1)}\widetilde{s}_k(t-1).
\end{aligned}
\tag{6.32}
$$

Using Equation (6.31), the remaining term of the cost function can be written as

$$
\begin{aligned}
C_p(s_k(t)) &= \mathrm{E}\left[-\log p(s_k(t)|\mathbf{s}(t-1), \boldsymbol{\theta})\right] \\
&= \frac{1}{2}\log(2\pi) + \overline{v}_{m_k} + \frac{1}{2}\alpha_k(t)\exp(2\widetilde{v}_{m_k} - 2\overline{v}_{m_k}).
\end{aligned}
\tag{6.33}
$$

6.2.2 Optimising the cost function

The most difficult part in optimising the cost function for the NSSM is updating the hidden states and the weights of the MLP networks. All the hyperparameters can be handled in exactly the same way as in the CDHMM case presented in Section 6.1.2 but ignoring the additional weights caused by the HMM state probabilities.

Updating the states and the weights is carried out in two steps. First the value of the cost function is evaluated using the current estimates for all the variables. This is called forward computation because it consists of a forward pass through the MLP networks.

The second, backward computation step consists of evaluating the partial derivatives of the C_p part of the cost function with respect to the different parameters. This can be done by moving backward in the network, starting from the outputs and proceeding toward the inputs. The standard back-propagation calculations are done in the same way. In our case, however, all the parameters are described by their own posterior distributions, which are characterised by their means and variances. The cost function is very different from the standard back-propagation and the learning is unsupervised. This means that all the calculation formulas are different.

Updating the network weights

Assume θ_i is a weight in one of the MLP networks and we have evaluated the partial derivatives $\partial C_p / \partial \overline{\theta}_i$ and $\partial C_p / \partial \widetilde{\theta}_i$. The variance $\widetilde{\theta}_i$ is easy to update with a fixed point update rule derived by setting the derivative to zero

$$0 = \frac{\partial C}{\partial \widetilde{\theta}_i} = \frac{\partial C_p}{\partial \widetilde{\theta}_i} + \frac{\partial C_q}{\partial \widetilde{\theta}_i} = \frac{\partial C_p}{\partial \widetilde{\theta}_i} - \frac{1}{2\widetilde{\theta}_i} \Rightarrow \widetilde{\theta}_i = \left(2 \frac{\partial C_p}{\partial \widetilde{\theta}_i} \right)^{-1}. \tag{6.34}$$

By looking at the form of the cost function for Gaussian terms, we can find an approximation for the second derivatives with respect to the means as [34]

$$\frac{\partial^2 C}{\partial \overline{\theta}_i^2} \approx 2 \frac{\partial C_p}{\partial \widetilde{\theta}_i} = \frac{1}{\widetilde{\theta}_i}. \tag{6.35}$$

This allows using an approximate Newton's iteration to update the mean

$$\overline{\theta}_i \leftarrow \overline{\theta}_i - \frac{\partial C}{\partial \overline{\theta}_i} \left(\frac{\partial^2 C}{\partial \overline{\theta}_i^2} \right)^{-1} \approx \overline{\theta}_i - \frac{\partial C}{\partial \overline{\theta}_i} \widetilde{\theta}_i. \tag{6.36}$$

There are some minor corrections to these update rules as explained in [34].

Updating the hidden states

The basic setting for updating the hidden states is the same as above for the network weights. The correlations between consecutive states cause some changes to the formulas and require new ones for adaptation of the correlation coefficients. All the feedforward computations use the marginal variances $\widetilde{s}_k(t)$ which are not actual variational parameters. This affects the derivatives with respect to the other parameters of the state distribution. Let us use the notation $C_p(\widetilde{s}_k(t))$ to mean that the C_p part of the cost function is considered to be a function of the intermediate variables $\widetilde{s}_k(1), \dots, \widetilde{s}_k(t)$ in addition to the variational parameters. This and Equation (5.48) yield following rules for evaluating the derivatives of the true cost function:

$$\frac{\partial C}{\partial \mathring{s}_k(t)} = \frac{\partial C_p(\widetilde{s}_k(t))}{\partial \widetilde{s}_k(t)} \frac{\partial \widetilde{s}_k(t)}{\partial \mathring{s}_k(t)} + \frac{\partial C_q}{\partial \mathring{s}_k(t)} = \frac{\partial C_p(\widetilde{s}_k(t))}{\partial \widetilde{s}_k(t)} - \frac{1}{2\mathring{s}_k(t)} \tag{6.37}$$

$$\begin{aligned} \frac{\partial C}{\partial \breve{s}_k(t-1,t)} &= \frac{\partial C_p(\widetilde{s}_k(t))}{\partial \breve{s}_k(t-1,t)} + \frac{\partial C_p(\widetilde{s}_k(t))}{\partial \widetilde{s}_k(t)} \frac{\partial \widetilde{s}_k(t)}{\partial \breve{s}_k(t-1,t)} \\ &= \frac{\partial C_p(\widetilde{s}_k(t))}{\partial \breve{s}_k(t-1,t)} + 2\frac{\partial C_p(\widetilde{s}_k(t))}{\partial \widetilde{s}_k(t)} \breve{s}_k(t-1,t)\widetilde{s}_k(t-1). \end{aligned} \tag{6.38}$$

The term $\partial C_p(\widetilde{s}_k(t))/\partial \widetilde{s}_k(t)$ in the above equations cannot be evaluated directly, but requires again the use of new intermediate variables. This leads to the recursive formula

$$\begin{aligned} \frac{\partial C_p(\widetilde{s}_k(t))}{\partial \widetilde{s}_k(t)} &= \frac{\partial C_p(\widetilde{s}_k(t+1))}{\partial \widetilde{s}_k(t)} + \frac{\partial C_p(\widetilde{s}_k(t+1))}{\partial \widetilde{s}_k(t+1)} \frac{\partial \widetilde{s}_k(t+1)}{\partial \widetilde{s}_k(t)} \\ &= \frac{\partial C_p(\widetilde{s}_k(t+1))}{\partial \widetilde{s}_k(t)} + \frac{\partial C_p(\widetilde{s}_k(t+1))}{\partial \widetilde{s}_k(t+1)} \breve{s}_k^2(t,t+1). \end{aligned} \tag{6.39}$$

The terms $\partial C_p(\widetilde{s}_k(t+1))/\partial \widetilde{s}_k(t)$ are now the ones that can be evaluated with the backward computations through the MLPs as usual.

The term $\partial C_p(\widetilde{s}_k(t))/\partial \breve{s}_k(t-1,t)$ is easy to evaluate from Equation (6.33), and it gives

$$\frac{\partial C_p(\widetilde{s}_k(t))}{\partial \breve{s}_k(t-1,t)} = -\frac{\partial g_k(\mathbf{s}(t-1))}{\partial s_k(t-1)} \widetilde{s}_k(t-1) \exp(2\widetilde{v}_{m_k} - 2\overline{v}_{m_k}). \tag{6.40}$$

Equations (6.38) and (6.40) yield a fixed point update rule for $\breve{s}_k(t-1,t)$:

$$\breve{s}_k(t-1,t) = \frac{\partial g_k(\mathbf{s}(t-1))}{\partial s_k(t-1)} \exp(2\widetilde{v}_{m_k} - 2\overline{v}_{m_k}) \left(2\frac{\partial C_p(\widetilde{s}_k(t))}{\partial \widetilde{s}_k(t)} \right)^{-1}. \tag{6.41}$$

The result depends, for instance, on $\breve{s}_k(t,t+1)$ through Equation (6.39), so the updates must be done in the order starting from the last and proceeding backward in time.

The fixed point update rule of the variances $\mathring{s}_k(t)$ can be solved from Equation (6.37):

$$\mathring{s}_k(t) = \left(2\frac{\partial C_p(\widetilde{s}_k(t))}{\partial \widetilde{s}_k(t)} \right)^{-1}. \tag{6.42}$$

The update rule for the means is similar to that of the weights in Equation (6.36) but it includes a correction which tries to compensate the simultaneous updates of the sources. The correction is explained in detail in [60].

6.2.3 Learning procedure

At the beginning of the learning for a new data set, the posterior means of the network weights are initialised to random values and the variances to small constant values. The original data is augmented with delay coordinate embedding, which was presented in Section 2.1.4, so that it consists of multiple time-shifted copies. The hidden states are initialised with a *principal component* (PCA) [27] projection of the augmented data. It is also used in training at the beginning of the learning.

The learning procedure of the NSSM consists of sweeps. During one sweep, all the parameters of the model are updated as outlined above. There are, however, different phases in learning so that not all the parameters are updated at the very beginning. These phases are summarised in Table 6.1.

Table 6.1: The different phases of NSSM learning.

Sweeps	Updates
0–50	Only the weights of the MLPs are updated, hidden states and hyperparameters are kept fixed at their initial values.
50–100	Only the weights and the hidden states are updated, hyperparameters are still kept fixed.
100–1000	Everything is updated using the augmented data.
1000	The original data is restored and the parts of the observation network $\mathbf{f}$ corresponding to the extra data are pruned away.
1000–10000	Everything is updated using the original data.

6.2.4 Continuing learning with new data

Continuing the learning process with the old model but new data requires initial estimates for the new hidden states. If the new data is a direct continuation of the old, the predictions of the old states provide a reasonable initial estimate for the new ones and the algorithm can continue the adaptation from there.

If the new data forms an entirely separate sequence, the problem is more difficult. Knowing the model, we can still do much better than starting at random or using the same initialisation as in the very beginning.

One way to find the estimates is to use an auxiliary MLP network to model the inverse of the observation mapping $\mathbf{f}$ [59]. This MLP can be trained using standard supervised back-propagation with the estimated means of $\mathbf{s}(t)$ and $\mathbf{x}(t)$ as training set. Their roles are of course inverted so that $\mathbf{x}(t)$ are the inputs and $\mathbf{s}(t)$ the outputs. The auxiliary MLP cannot give perfect estimates for the states $\mathbf{s}(t)$, but they can usually be adapted very quickly by using the standard learning algorithm to update only the hidden states.

6.3 Learning algorithm for the switching model

In this section, the learning procedure for the switching model is presented.

6.3.1 Evaluating and optimising the cost function

The switching model is a combination of the two models as shown in Section 5.3. Therefore most of the terms of the cost function are exactly as they were in the individual models. The only difference is in the term $C_p(s_k(t)) = \mathrm{E}[-\log p(s_k(t)|\cdots)]$. The term $\alpha_k(t)$ in Equation (6.31) changes to $\alpha_{k,i}(t)$ for the case $M_t = i$ and has the value

$$\begin{aligned}\alpha_{k,i}(t) &= \mathrm{E}\left[(s_k(t) - g_k(\mathbf{s}(t-1)) - m_{M_i,k})^2\right] \\ &= (\overline{s}_k(t) - \overline{g}(\mathbf{s}(t-1)) - \overline{m}_{M_i,k})^2 + \widetilde{s}_k(t) + \widetilde{m}_{M_i,k} \\ &\quad + \widetilde{g}_k(\mathbf{s}(t-1)) - 2\breve{s}_k(t-1,t)\frac{\partial g_k(\mathbf{s}(t-1))}{\partial s_k(t-1)}\widetilde{s}_k(t-1).\end{aligned} \tag{6.43}$$

With this result the expectation of Equation (6.33) involving the source prior becomes

$$\begin{aligned}C_p(\boldsymbol{S}) &= \mathrm{E}\left[-\log p(s_k(t)|\mathbf{s}(t-1), M_t, \boldsymbol{\theta})\right] \\ &= \frac{1}{2}\log(2\pi) + \sum_{i=1}^{N} q(M_t = i)\left(\overline{v}_{M_i,k} + \frac{1}{2}\alpha_{k,i}(t)\exp(2\widetilde{v}_{M_i,k} - 2\overline{v}_{M_i,k})\right).\end{aligned} \tag{6.44}$$

The update rules will mostly stay the same except that the values of the parameters of v_{m_k} must be replaced with properly weighted averages of the corresponding parameters of $v_{M_i,k}$. The HMM prototype vectors will be taught using $\mathbf{s}(t) - \mathbf{g}(\mathbf{s}(t-1))$ as the data.

6.3.2 Learning procedure

The progress of learning in the switching NSSM is almost the same as in the plain NSSM. The parameters are updated in similar sweeps and the data are used in exactly the same way.

The HMM prototype means are initialised to have relatively small random means and small constant variances. The prototype variances are initialised to suitable constant values.

The phases in learning the switching model are presented in Table 6.2.

Table 6.2: The different phases of switching NSSM learning.

Sweeps	Updates
0–50	Only the weights of the MLPs and the HMM states are updated, continuous hidden states, HMM prototypes and hyperparameters are kept fixed at their initial values.
50–100	Only the weights and both the hidden states are updated, HMM prototypes and hyperparameters are still kept fixed.
100–500	Everything but the HMM prototypes is updated.
500–1000	Everything is updated using the augmented data.
1000	The original data is restored and the parts of the observation network **f** corresponding to the extra data are pruned away.
1000–10000	Everything is updated using the original data.

In each sweep of the learning algorithm, the following computations are performed:

- The distributions of the outputs of the MLP networks **f** and **g** are evaluated as presented in Appendix B.
- The HMM state probabilities are updated as in Equation (6.14).
- The partial derivatives of the cost function with respect to the weights and inputs of the MLP networks are evaluated by inverting the computations of Appendix B and using Equations (6.37)–(6.39).

- The parameters for the continuous hidden states $\mathbf{s}(t)$ are updated using Equations (6.36), (6.41) and (6.42).
- The parameters of the MLP network weights are updated using Equations (6.34) and (6.36).
- The HMM output parameters are updated using Equations (6.20) and (6.22), and the results from solving Equations (6.23)–(6.24).
- The hyperparameters of the HMM are updated using Equations (6.16) and (6.17).
- All the other hyperparameters are updated using similar procedure as with the HMM output parameters.

6.3.3 Learning with known state sequence

Sometimes we want to use the switching NSSM to model data we already know something about. With speech data, for instance, we may know the "correct" sequence of phonemes in the utterance. This does not mean that learning the HMM part would be unnecessary. The correct segmentation requires determining the times of transitions between the states. Now only the states the model has to pass and their order are given.

Such problems can be solved by estimating the HMM states for a modified model, namely the one that only allows transitions in the correct sequence. These probabilities can then be transformed back to the true state probabilities for the adaptation of the other model parameters. The forward–backward procedure must also be modified slightly as the first and the last state of a sequence are now known for sure.

When the correct state sequences are known, the different orderings of HMM states are no longer equivalent. Therefore the HMM output distribution parameters can, and actually should, all be initialised to zeros. Random initialisation could make the output model for certain state very different from the true output thus making the learning much more difficult.

Chapter 7

Experimental results

A series of experiments was conducted to verify that the switching NSSM model and the learning algorithm derived in Chapters 5 and 6 really work. The results of these experiments are presented in this chapter.

All the experiments were done with different parts of one large data set of speech. In Section 7.1, the data set used and some of its properties are introduced. In Section 7.2, the developed switching NSSM is compared with standard HMM, NSSM and NFA models. The results of a segmentation experiment are presented in Section 7.3.

7.1 Speech data

The data collection that was used in the experiments consisted of individual Finnish words, spoken by 59 different speakers. The data has been collected at the Laboratory of Computer and Information Science. A detailed description of the data and its collection process can be found in Vesa Siivola's Master's thesis [54].

Due to the complexity of the algorithms used, only a very small fraction of the whole collection was ever used. The time needed for learning increases linearly as the amount of data increases and with more data even a single experiment would have taken months to complete. The data sets used in the experiments were selected by randomly taking individual words from the whole collection. In a typical experiment, where the used data set consisted of only a few dozen words, this meant that practically every word in the set was spoken by a different person.

7.1.1 Preprocessing

The preprocessing performed to turn the digitised speech samples to the observation vectors for the algorithms was as follows:

1. The signal was high-pass filtered to emphasise the important higher frequencies. This was done with a first order FIR filter having the transfer function $H(z) = 1 - 0.95z^{-1}$.

2. A 256-point Fourier transform with Hamming windowing was calculated for short overlapping segments. The overlapping part of two consecutive segments consisted of half of the segments.

3. The frequencies were transformed to *Mel*-scale to emphasise the important features for understanding the speech. This gave a 30 component vector for each segment.

4. The logarithm of the energies on the *Mel*-scale was used as observations.

These steps form a rather standard preprocessing procedure for speech recognition [54, 33].

The Mel-scale of frequencies has been designed to model the frequency response of the human ear. The scale is constructed by asking a naïve listener when she found the heard sound to have double or half of the frequency of a reference tone. The resulting scale is close to linear at frequencies below 1000 Hz and nearly logarithmic above that [54].

Figure 7.1 shows an example of what the preprocessed data looks like.

7.1.2 Properties of the data set

One distinctive property of the data is that it is not very continuous. This is due to the bad frequency resolution of the relatively short time Fourier transform used in the preprocessing.

The same data set was used with the static NFA model in [25]. The part used in these experiments consisted of spectrograms of 24 individual words, spoken by 20 different speakers. The preprocessed data consisted of 2547 spectrogram vectors with 30 components.

For studying the dimensionality of the data, linear and nonlinear factor analysis were applied to the data. The results are shown in Figure 7.2. All the NFA

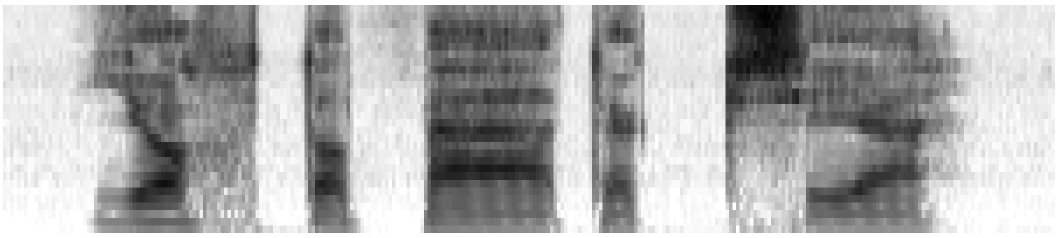


Figure 7.1: An example of the preprocessed spectrogram of a speech segment. Time increases from left to right and frequency from down to up. White areas correspond to low energy of the signal and dark areas to high energy. The word in the segment is “JOHTOPÄÄTÖKSIÄ”, meaning “conclusions”. Every letter in the written word corresponds to one phoneme in speech. The silent areas in the middle correspond to the consonants t, p, t and k, thus revealing the segmentation of the utterance into phonemes.

experiments used an MLP network with 30 hidden neurons. The data manifold is clearly nonlinear, because nonlinear factor analysis is able to explain it equally well with fewer components than linear factor analysis. The difference is especially clear when the number of components is relatively small. Even though the analysis only uses static models, it can be used to estimate a lower bound for the number of continuous hidden states used in the experiments with dynamical models.

A small segment of the original data and its reconstructions with eight nonlinear and linear components are shown in Figure 7.3. The reconstructed spectrograms are somewhat smoother than the original ones. Still, all the discriminative features of the original spectrum are well preserved in the nonlinear reconstruction. This means that the dropped components mostly correspond to noise. The linear reconstruction is not as good, especially at the beginning.

The extracted nonlinear factors, rotated with linear ICA, are shown in Figure 7.4. They seem rather smooth so it seems plausible that the dynamic models would be able to model the data better. The representation of the data given by the nonlinear factors seems, however, somewhat more difficult to interpret. It is rather difficult to see how the different factors affect the predicted outputs.

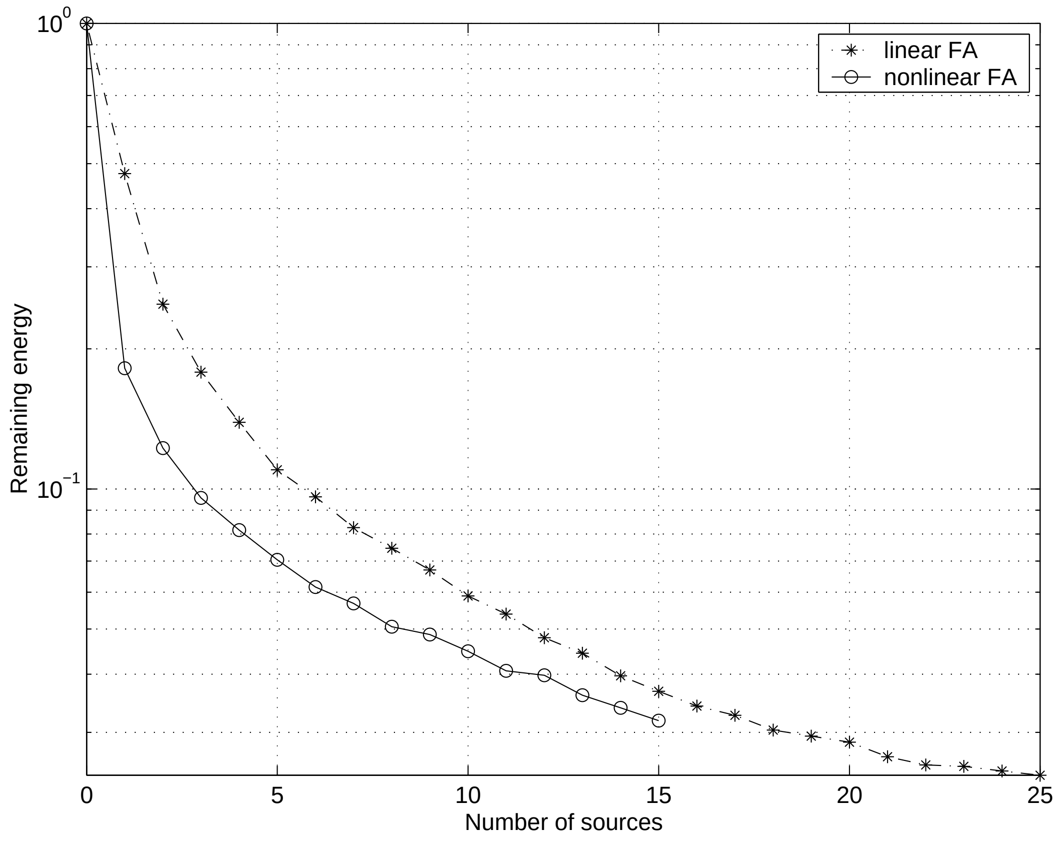


Figure 7.2: The remaining energy of the speech data as a function of the number of extracted components using linear and nonlinear factor analysis.

7.2 Comparison with other models

In this section, a comparative test between the switching NSSM and some other standard models is presented. The goodness of the models is measured with the *model evidence*, i.e. the likelihood $p(X|\mathcal{H}_i)$ for given data set X and model $\mathcal{H}_i$. The theory behind this comparison is presented in Section 3.2.1. The values of the evidence are evaluated using the variational approximation given by ensemble learning.

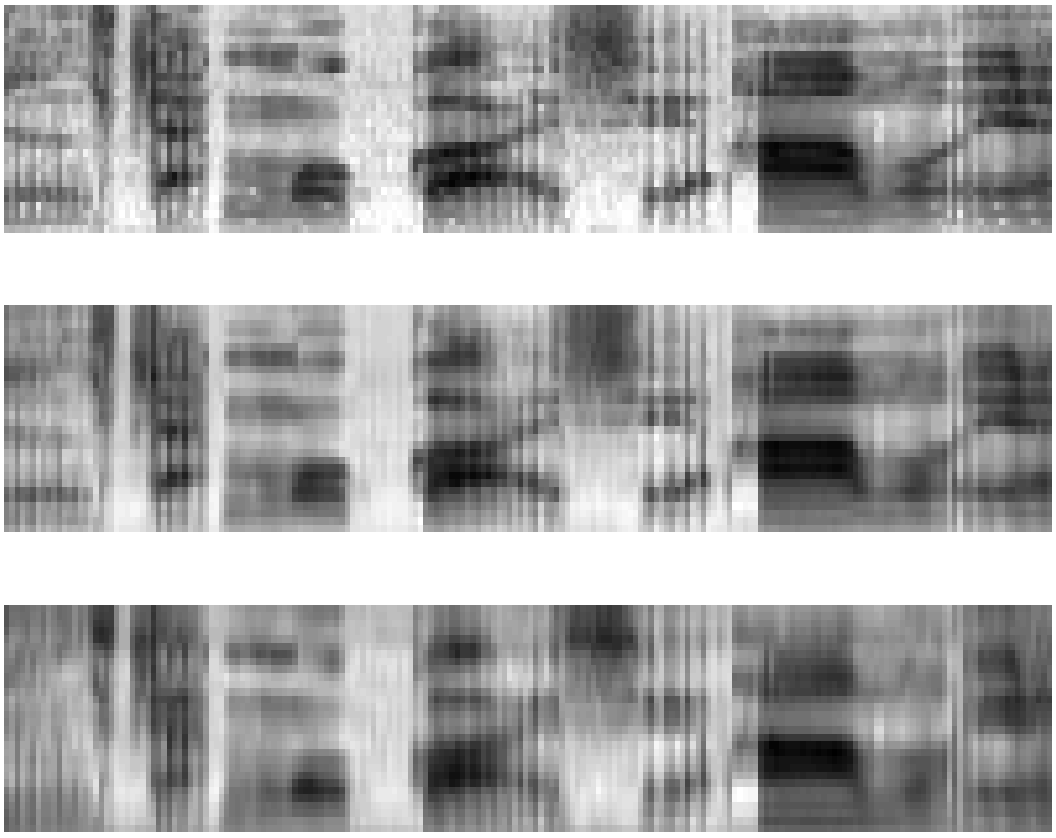


Figure 7.3: A short fragment of the data used in the experiment. The first subfigure shows the original data, the second shows the reconstruction from 8 nonlinear components and the last shows the reconstruction from 8 linear components. Both the models used were static and did not use any temporal information on the signals. The results would have been exactly the same for any permutation of the data vectors.

7.2.1 The experimental setting

All the models used the same preprocessed data set as in Section 7.1.2. The individual words were processed separately in the preprocessing and all the dynamical models were instructed to treat each word individually, i.e. not to make predictions across word boundaries.

The models used in the comparison were:

- A continuous density HMM with Gaussian mixture observation model.

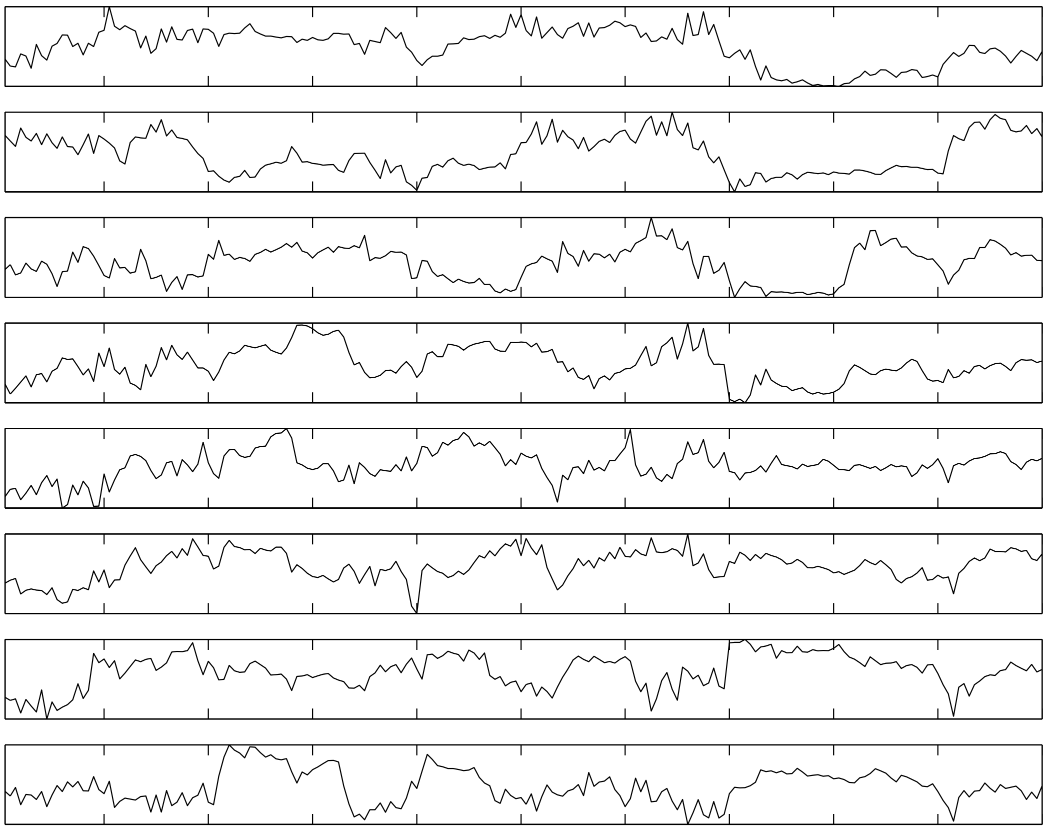


Figure 7.4: Extracted NFA factors corresponding to the data fragment in Figure 7.3, rotated with linear ICA.

This was a simple extension to the model presented in Section 5.1 that replaced the Gaussian observation model with mixtures-of-Gaussians. The number of Gaussians was the same for all the states but it was optimised by running the algorithm with several values and using the best result. The model was initialised with sufficiently many states and unused extra states were pruned.

- The nonlinear factor analysis model, as presented in Section 4.2.4. The model had 15 dimensional factors and 30 hidden neurons in the observation MLP network.
- The nonlinear SSM, as presented in Section 5.2. The model had 15 dimensional state-space and 30 hidden neurons in both MLP networks.
- The switching NSSM, as presented in Section 5.3. The model was essen-

tially a combination of the HMM and NSSM models except that it used Gaussian observation model for the HMM.

The parameters of the HMM priors for the initial distribution $u^{(\pi)}$ and the transition matrix $u^{(A)}$ were all set to ones. This corresponds to a flat, noninformative prior. The choice does not affect the performance of the switching NSSM very much. The HMM, on the other hand, is very sensitive to the prior.

The data used with the plain HMM was additionally decorrelated with *principal component analysis* (PCA) [27]. This improved the performance a lot compared to the situation without the decorrelation, as the prototype Gaussians were restricted to be uncorrelated. The other algorithms can include the same transformation to the output mapping so it was not necessary to do it by hand. Using the nondecorrelated data has the advantage that it is "human readable" whereas the decorrelated data is much more difficult to interpret.

7.2.2 The results

The results reached by different models are summarised in Table 7.1. The static NFA model gets the worst score in describing the data. It is a little faster than the NSSM and switching NSSM but significantly slower than the HMM. The HMM is a little better and it is clearly the fastest of the algorithms. The NSSM is significantly better than the HMM but takes quite a lot more time. The switching NSSM is a clear winner in describing the data but it is also the slowest of all the algorithms. The difference in speeds of NSSMs with and without switching is relatively small.

Table 7.1: The results of the model comparison experiment. The second column contains the values of the ensemble learning cost function attained. Lower values are better. The values translate to probabilities as $p(X|\mathcal{H}) \approx e^{-C}$. The third column contains a rough estimate on the time needed to run one simulation with Matlab on a single relatively fast RISC processor.

Model	Cost function value	Time needed
NFA	111 041	A few days
HMM	104 654	About an hour
NSSM	90 955	About a week
Switching NSSM	82 410	More than a week

The simulations with NFA, NSSM and switching NSSM were run using a 15 dimensional latent space. The results would probably have been slightly better with larger dimensionality, but unfortunately the current optimisation algorithm used for the models is somewhat unstable above that limit. Optimisation of the structures of the MLP networks would also have helped, but it would have taken too much time to be practical. The present results are thus the ones attained by taking the best of a few simulations with the same fixed structure but different random initialisations.

7.3 Segmentation of annotated data

In this section, an experiment dealing with data segmentation is presented. The correct phoneme sequence for each segment of the speech data is known but the correct segmentation, i.e. the times of the transitions between the phonemes, is not. Therefore it is reasonable to see whether the model can learn to find the correct segmentation.

The HMM used in this experiment had four states for each phoneme of the data. These states were linked together in a chain. Transitions from "outside" were allowed only to the first state and then forward in the chain until the last internal state of the phoneme was reached. This is a standard procedure in speech recognition to model the duration of the phonemes.

7.3.1 The training procedure

In this experiment, the switching NSSM is used in a partially supervised mode, i.e. it is trained as explained in Section 6.3.3. Using only this modification does not, however, lead to good results. The computational complexity of the NSSM learning allows using only a couple of thousands of samples of data. With the preprocessing used in this work, this means that the NSSM must be trained with only a few dozen words. This training material is not at all enough to learn models for all the phonemes, as many of them appear only a few times in the training data set.

To circumvent this problem, the training of the model was split into several phases. In the first phase, the complete model was trained with a data set of 23 words forming some 5069 sample vectors. This should be enough for the NSSM part to find a reasonable representation for the data. The number of sweeps for this training was 10000. After this step no NSSM parameters were adapted any more.

The number of words in the training set of the first phase is small when compared to the data set of the previous experiment. This is due to inclusion of significant segments of silence to the samples. It is important for the model to also learn a good model for silence, since in real life speech is always embedded into silence or plain background noise.

In the second phase, the training data set was changed to a new one for continuing training the HMM. The new data set consisted of 786 words forming 100064 sample vectors. The continuous hidden states for the new data were initialised with an auxiliary MLP as presented in Section 6.2.4. Then the standard learning algorithm was used for 10 sweeps to adapt only the continuous states.

Updating just the HMM and leaving the NSSM out saved a significant amount of time. As Table 7.1 shows, HMM training is about two orders of magnitude faster than NSSM training. Using just the HMM part takes about 20 % of the time needed by the complete switching NSSM. The rest of the improvement is due to the fact that the HMM training converges with far fewer iterations. These savings allowed using a much larger data set to efficiently train the HMM part.

7.3.2 The results

After the first phase of training with the small data set, the segmentations done by the algorithm seem pretty random. This can be seen from Figure 7.5. The model has not even learnt how to separate the speech signal from the silence at the beginning and end of the data segments.

After the full training the segmentations seem rather good, as Figures 7.6 and 7.7 show. This is a very encouraging result, considering that the segmentations are performed using only the innovation process (the last subfigure) of the NSSM which consists mostly of the leftovers of the other parts of the model. The results should be significantly better with a model that gives the HMM a larger part in predicting the data.

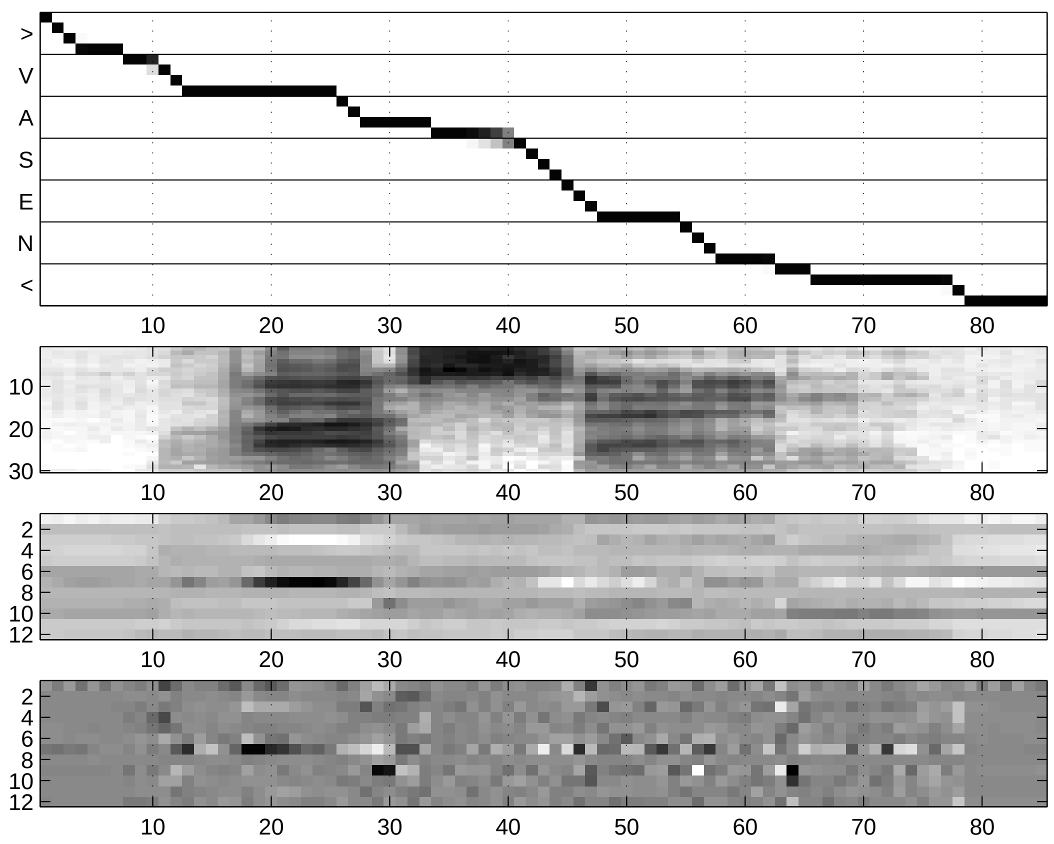


Figure 7.5: An example of the segmentation given by the algorithm after the first phase of learning. The states '>' and '<' correspond to silence at the beginning and the end of the utterance. The first subfigure shows the marginal probabilities of the HMM states for each sample. The second subfigure shows the data, the third shows the continuous hidden states $\mathbf{s}(t)$ and the last shows the innovation processes $\mathbf{s}(t) - \mathbf{g}(\mathbf{s}(t-1))$. The HMM does its segmentation solely based on the values of the innovation process, i.e. the last subfigure. The word in the figure is "VASEN" meaning "left".

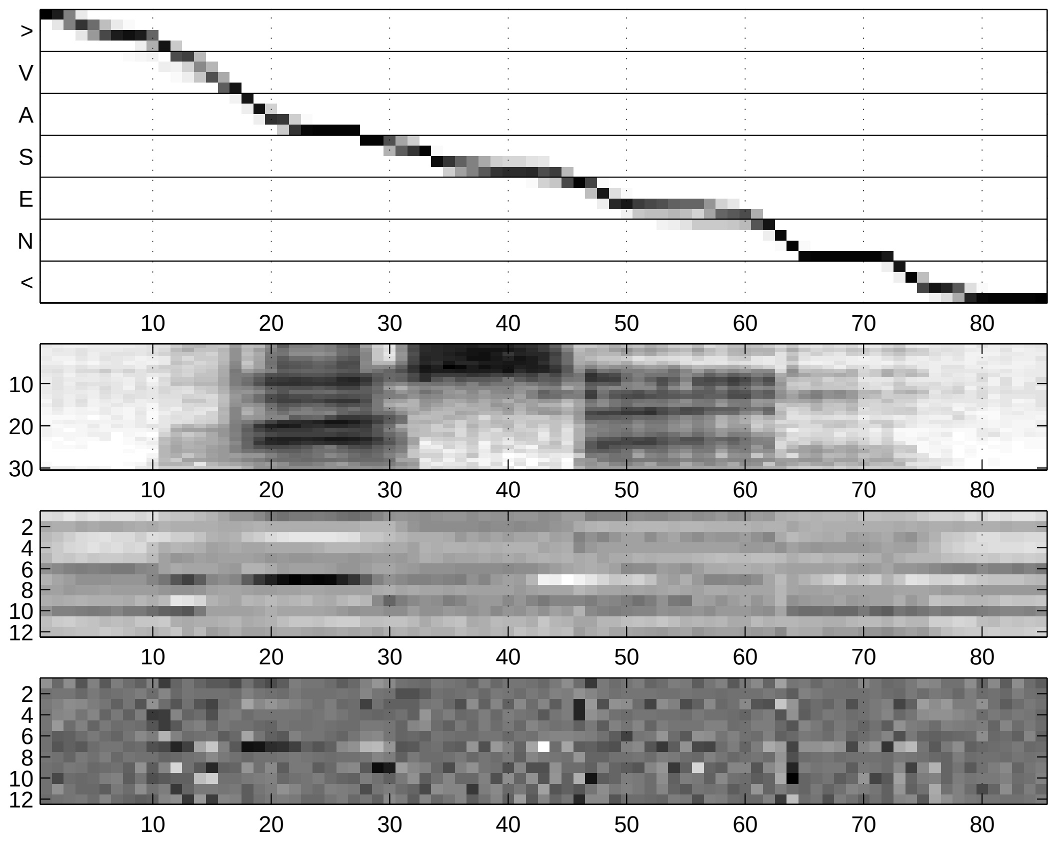


Figure 7.6: An example of the segmentation given by the algorithm after complete learning. The data and the meanings of the different parts of the figure are the same as in Figure 7.5. The results are significantly better though not yet quite perfect.

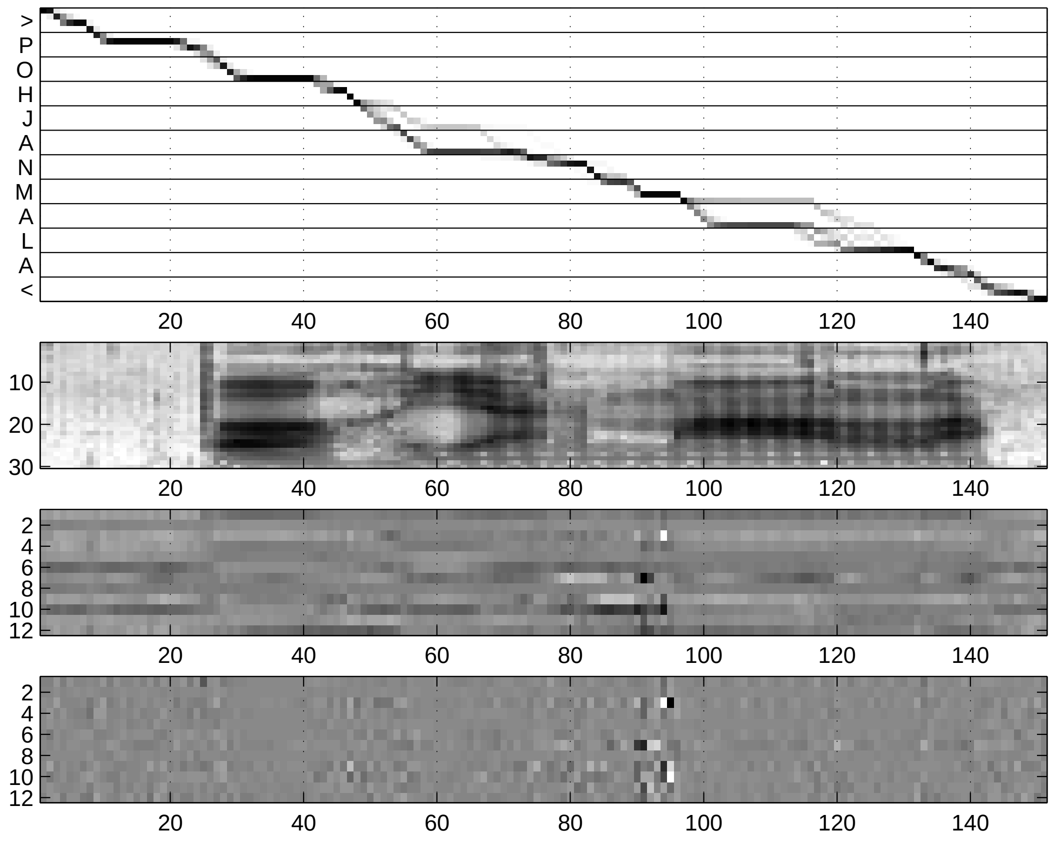


Figure 7.7: Another example of segmentation given by the algorithm after complete learning. The meanings of the different parts of the figure are the same as in Figures 7.5 and 7.6. The figure illustrates the segmentation of a longer word. The result shows several relatively probable paths, not just one as in the previous figures. The word in the figure is "POHJANMAALLA". The double phonemes are treated as one in the segmentation.

Chapter 8

Discussion

In this work, a Bayesian switching nonlinear state-space model (switching NSSM) and a learning algorithm for its parameters were developed. The switching model combines two dynamical models, a hidden Markov model (HMM) and a nonlinear state-space model (NSSM). The HMM models long-term behaviour of the data and controls the NSSM which describes the short-term dynamics of the data. In order to be used in practice, the switching NSSM is, like any other model, needs an efficient method to learn its parameters. In this work, the Bayesian approximation method called ensemble learning was used to derive a learning algorithm for the model.

The requirements of the learning algorithm set some limitations for the structure of the model. The learning algorithm for the NSSM which was used as the starting point for this work was computationally intensive. Therefore the additions needed for the switching model had to be designed very carefully to avoid making the model computationally intractable. Most existing switching state-space model structures use entirely different dynamical models for each state of the HMM. Such an approach would have resulted in a more powerful model, but the computational burden would have been too great to be practical. Therefore the developed switching model uses the same NSSM for all the HMM states. The HMM is only used to model the prediction errors of the NSSM.

The ensemble learning based learning algorithm seems well suited for the switching NSSM. This kind of models usually suffer from the problem that the exact posterior of the hidden states is an exponentially growing mixture of Gaussian distributions. The ensemble learning approach solves this problem elegantly by finding the best approximate posterior consisting only of a single Gaussian distribution. It would also be possible to approximate the posterior with a mixture of a few Gaussian distributions instead of the single Gaussian. This would, however, increase the computational burden quite a lot while achieving little gain.

Despite the suboptimal model structure, the switching model performs remarkably well in the experiments. It outperforms all the other tested models in the speech modelling problem by a large margin. Additionally, the switching model yields a segmentation of the data to different discrete dynamical states. This allows using the model for many different segmentation tasks.

For a segmentation problem, the closest competitor of the switching NSSM is the plain HMM. The speech modelling experiment shows that the switching NSSM is significantly better in modelling the speech data than the HMM. This means that the switching model can get more out of the same data. The greatest problem of the switching model is the computational cost. It is approximately two orders of magnitude slower to train than the HMM. The difference in actually using the fully trained models should, however, be smaller. Much of the difference in training times is caused by the fact that the weights of the MLP networks take very many iterations of the training algorithm to converge. When the system is in operation and the MLPs are not trained, the other parts needed for recognition purposes converge much more quickly. The usage of the switching model in such a recognition system has not been thoroughly studied. Therefore it might be possible to find ways of optimising the usage of the model to make it comparable with the HMM.

In the segmentation experiment, the switching model learnt the desired segmentation of an annotated data set of individual words of speech to different phonemes. However, the initial experiments on using the model for recognition, i.e. finding the segmentation without the annotation, are not as promising. The poor recognition performance is probably due to the fact that the HMM part of the model only uses the prediction error of the continuous NSSM. When the state of the dynamics changes, the prediction error grows. Thus it is easy for the HMM to see that something is changing but not how it is changing. The HMM would have to have more influence on the description of the dynamics of the data to know which state the system is going to.

The design of the model was motivated by computational efficiency and the resulting algorithm seems successful in that sense. The learning algorithm for the switching model is only about 25 % slower than the corresponding algorithm for the NSSM. This could probably still be improved as Matlab is far from an optimal programming environment for HMM calculations. The forward and backward calculations of the MLP networks, which require evaluating the Jacobian matrix of the network at each input point, constitute the slowest part of the NSSM training. Those parts remain unchanged in the switching model and still take most of the time.

There are at least two important lines of future work: giving the model more expressive power by making better use of the HMM part and optimising the speed of the NSSM part.

Improving the model structure to give the HMM a larger role would be important to make the model work in recognition problems. The HMM should be given control over at least some of the parameters of the dynamical mapping. It is, however, difficult to do this without increasing the computational burden of using the model. Time usually helps in this respect as the computers become faster. Joining the two component models together more tightly would also make the learning algorithm even more complicated. Nevertheless, it might be possible to find a better compromise between the computational burden and the power of the model.

Another way to help with the essentially same problem would be optimising the speed of the NSSM. The present algorithm is very good in modelling the data but it is also rather slow. Using a different kind of a structure for the model might help with the speed problem. This would, however, probably require a completely new architecture for the model.

All in all, the nonlinear switching state-space models form an interesting field of study. At present, the algorithms seem computationally too intensive for most practical uses, but this is likely to change in the future.

Appendix A

Standard probability distributions

A.1 Normal distribution

The normal distribution, which is also known as the Gaussian distribution, is ubiquitous in statistics. The averages of identically distributed random variables are approximately normally distributed by the central limit theorem, regardless of their original distribution[16]. This section concentrates on the univariate normal distribution, as the general multivariate distribution is not needed in this thesis.

The probability density of the normal distribution is given by

$$p(x) = N(x;\ \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right). \tag{A.1}$$

The parameters of the distribution directly yield the mean and the variance of the distribution: $\mathrm{E}[x] = \mu$, $\mathrm{Var}[x] = \sigma^2$.

The multivariate case is very similar:

$$p(\mathbf{x}) = N(\mathbf{x};\ \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{2\pi}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}) \right) \tag{A.2}$$

where $\boldsymbol{\mu}$ is the mean vector and $\boldsymbol{\Sigma}$ the covariance matrix of the distribution. For our purposes it is sufficient to note that when the covariance matrix $\boldsymbol{\Sigma}$ is diagonal, the multivariate normal distribution reduces to a product of independent univariate normal distributions.

By the definition of the variance

$$\begin{aligned}\sigma^2 = \mathrm{Var}[x] &= \mathrm{E}[(x-\mu)^2] = \mathrm{E}[x^2 - 2x\mu + \mu^2] \\ &= \mathrm{E}[x^2] - 2\mu\,\mathrm{E}[x] + \mu^2 = \mathrm{E}[x^2] - \mu^2.\end{aligned} \tag{A.3}$$

This gives

$$\mathrm{E}[x^2] = \mu^2 + \sigma^2. \tag{A.4}$$

The negative differential entropy of the normal distribution can be evaluated simply as

$$\mathrm{E}[\log p(x)] = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\,\mathrm{E}\left[\frac{(x-\mu)^2}{\sigma^2}\right] = -\frac{1}{2}(\log(2\pi\sigma^2)+1). \tag{A.5}$$

Another important expectation for our purposes is [35]

$$\begin{aligned}\mathrm{E}[\exp(-2x)] &= \int \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \exp(-2x)dx \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \int \exp\left(-\frac{(x-\mu)^2 + 4x\sigma^2}{2\sigma^2}\right) dx \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \int \exp\left(-\frac{x^2 - 2\mu x + \mu^2 + 4x\sigma^2}{2\sigma^2}\right) dx \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \int \exp\left(-\frac{[x + (2\sigma^2 - \mu)]^2 + 4\mu\sigma^2 - 4(\sigma^2)^2}{2\sigma^2}\right) dx \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \int \exp\left(-\frac{[x + (2\sigma^2 - \mu)]^2}{2\sigma^2}\right) \exp\left(2\sigma^2 - 2\mu\right) dx \\ &= \exp\left(2\sigma^2 - 2\mu\right).\end{aligned} \tag{A.6}$$

A plot of the probability density function of the normal distribution is shown in Figure A.1.

A.2 Dirichlet distribution

The multinomial distribution is a discrete distribution which gives the probability of choosing a given collection of m items from a set of n items with repetitions and the probabilities of each choice given by $p_1, \dots, p_n$. These probabilities are the parameters of the multinomial distribution [16].

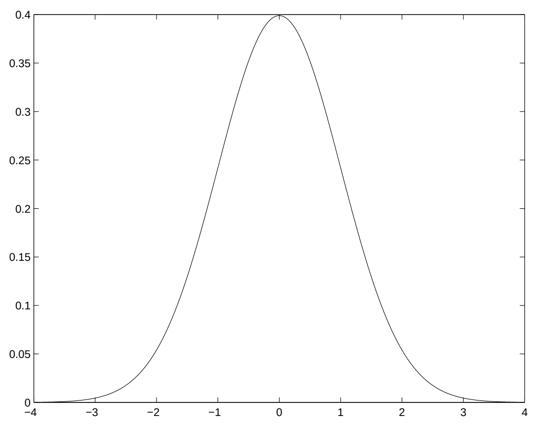


Figure A.1: Plot of the probability density function of the unit variance zero mean normal distribution $N(0, 1)$.

The Dirichlet distribution is the conjugate prior of the parameters of the multinomial distribution. The probability density of the Dirichlet distribution for variables $\mathbf{p} = (p_1, \ldots, p_n)$ with parameters $\mathbf{u} = (u_1, \ldots, u_n)$ is defined by

$$p(\mathbf{p}) = \text{Dirichlet}(\mathbf{p};\ \mathbf{u}) = \frac{1}{Z(\mathbf{u})} \prod_{i=1}^{n} p_i^{u_i - 1} \tag{A.7}$$

when $p_1, \ldots, p_n \geq 0; \sum_{i=1}^{n} p_i = 1$ and $u_1, \ldots, u_n > 0$. The parameters u_i can be interpreted as "prior observation counts" for events governed by p_i. The normalisation constant $Z(u)$ becomes

$$Z(\mathbf{u}) = \frac{\prod_{i=1}^{n} \Gamma(u_i)}{\Gamma(\sum_{i=1}^{n} u_i)}. \tag{A.8}$$

Let $u_0 = \sum_{i=1}^{n} u_i$. The mean and variance of the distribution are [16]

$$\text{E}[p_i] = \frac{u_i}{u_0} \tag{A.9}$$

and

$$\text{Var}[p_i] = \frac{u_i(u_0 - u_i)}{u_0^2(u_0 + 1)}. \tag{A.10}$$

When $u_i \to 0$, the distribution becomes noninformative. The means of all the p_i stay the same if all u_i are scaled with the same multiplicative constant. The variances will, however, get smaller as the parameters u_i grow. The pdfs of the Dirichlet distribution with certain parameter values are shown in Figure A.2.

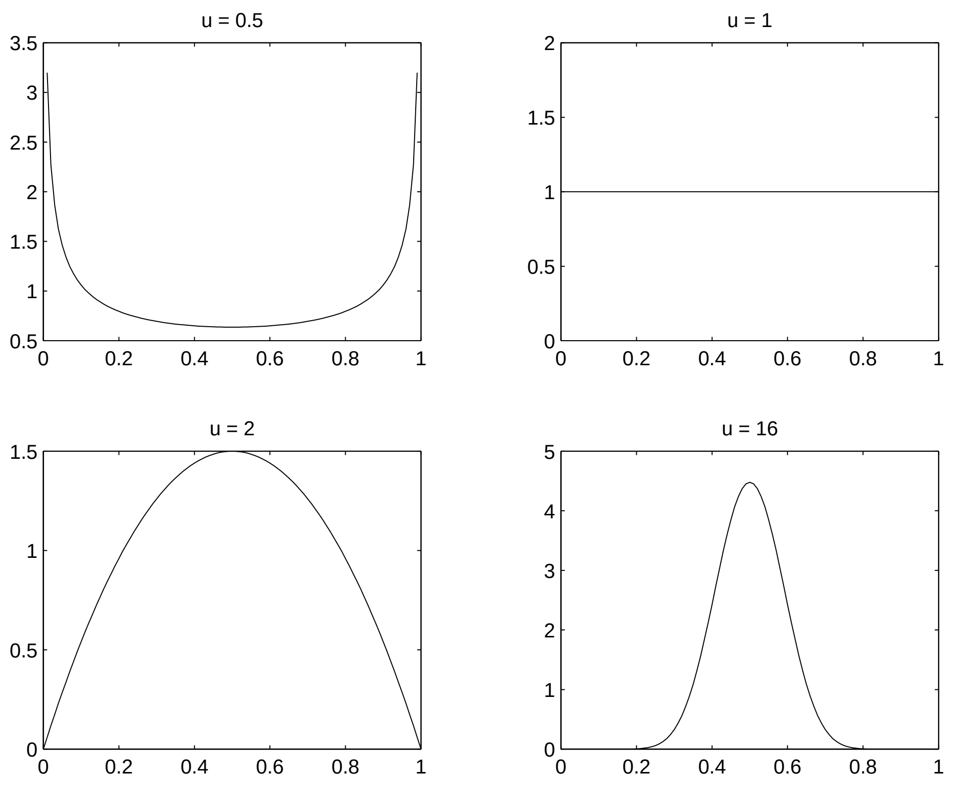


Figure A.2: Plots of one component of a two dimensional Dirichlet distribution. The parameters are chosen such that $u_1 = u_2 = u$ with the values for u shown above each individual image. Because both the parameters of the distribution are equal, the distribution of the other component will be exactly the same.

In addition to the standard statistics given above, using ensemble learning for parameters with Dirichlet distribution requires the evaluation of the expectation $\mathrm{E}[\log p_i]$ and the negative differential entropy $\mathrm{E}[\log p(\mathbf{p})]$.

The first expectation can be reduced to evaluating the expectation over a two dimensional Dirichlet distribution for

$$(p, 1-p) \sim \mathrm{Dirichlet}(u_i, u_0 - u_i) \tag{A.11}$$

which is given by the integral

$$\mathrm{E}[\log p_i] = \int_0^1 \frac{\Gamma(u_0)}{\Gamma(u_i)\Gamma(u_0 - u_i)} p^{u_i - 1}(1-p)^{u_0 - u_i - 1} \log p \, dp. \tag{A.12}$$

This can be evaluated analytically to yield

$$\mathrm{E}[\log p_i] = \Psi(u_i) - \Psi(u_0) \tag{A.13}$$

where $\Psi(x) = \frac{d}{dx} \ln(\Gamma(x))$ is also known as the digamma function.

By using this result, the negative differential entropy can be evaluated

$$\begin{aligned}
\mathrm{E}[\log p(\mathbf{p})] &= \mathrm{E}\left[\log\left(\frac{1}{Z(\mathbf{u})}\prod_{i=1}^{n} p_i^{u_i-1}\right)\right] \\
&= -\log Z(\mathbf{u}) + \sum_{i=1}^{n}(u_i - 1)\,\mathrm{E}[\log p_i] \qquad (A.14) \\
&= -\log Z(\mathbf{u}) + \sum_{i=1}^{n}(u_i - 1)[\Psi(u_i) - \Psi(u_0)]
\end{aligned}$$

Appendix B

Probabilistic computations for MLP networks

In this appendix, it is shown how to evaluate the distribution of the outputs of an MLP network $\mathbf{f}$, assuming the inputs $\mathbf{s}$ and all the network weights are independent and Gaussian. These equations are needed in evaluating the ensemble learning cost function of the Bayesian NSSM in Section 6.2.

The exact model for our single-hidden-layer MLP network $\mathbf{f}$ is

$$\mathbf{f}(\mathbf{s}) = \mathbf{B}\boldsymbol{\varphi}(\mathbf{A}\mathbf{s} + \mathbf{a}) + \mathbf{b}. \tag{B.1}$$

The structure is shown in Figure B.1. The assumed distributions for all the parameters, which are assumed to be independent, are

$$s_i \sim N(\overline{s}_i, \widetilde{s}_i) \tag{B.2}$$

$$A_{ij} \sim N(\overline{A}_{ij}, \widetilde{A}_{ij}) \tag{B.3}$$

$$B_{ij} \sim N(\overline{B}_{ij}, \widetilde{B}_{ij}) \tag{B.4}$$

$$a_i \sim N(\overline{a}_i, \widetilde{a}_i) \tag{B.5}$$

$$b_i \sim N(\overline{b}_i, \widetilde{b}_i). \tag{B.6}$$

The computations in the first layer of the network can be written as $y_i = a_i + \sum_j A_{ij} s_j$. Since all the parameters involved are independent, the mean

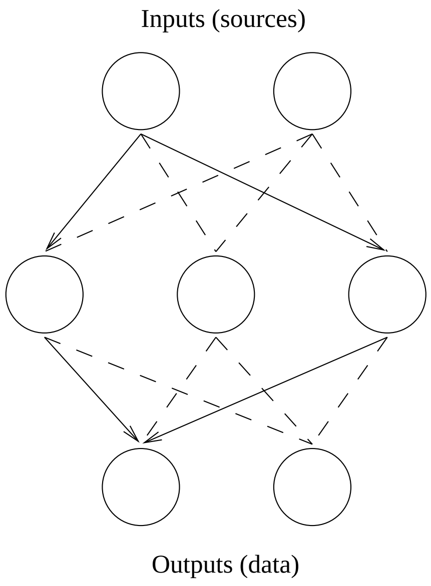


Figure B.1: The structure of an MLP network with one hidden layer.

and the variance of y_i are

$$\overline{y}_i = \overline{a}_i + \sum_j \overline{A}_{ij}\overline{s}_j \tag{B.7}$$

$$\widetilde{y}_i = \widetilde{a}_i + \sum_j \left[\overline{A}_{ij}^2\widetilde{s}_j + \widetilde{A}_{ij}\left(\overline{s}_j^2 + \widetilde{s}_j\right)\right]. \tag{B.8}$$

Equation (B.8) follows from the identity

$$\mathrm{Var}[\alpha] = \mathrm{E}[\alpha^2] - \mathrm{E}[\alpha]^2. \tag{B.9}$$

The nonlinear activation function is handled with a truncated Taylor series approximation about the mean $\overline{y}_i$ of the inputs. Using a second order approximation for the mean and first order for the variance yields

$$\overline{\varphi}(y_i) \approx \varphi(\overline{y}_i) + \frac{1}{2}\varphi''(\overline{y}_i)\widetilde{y}_i \tag{B.10}$$

$$\widetilde{\varphi}(y_i) \approx \left[\varphi'(\overline{y}_i)\right]^2 \widetilde{y}_i. \tag{B.11}$$

The reason why these approximations are used is that they are the best ones that can be expressed in terms of the input mean and variance.

The computations of the output layer are given by $f_i(\mathbf{s}) = b_i + \sum_j B_{ij}\varphi(y_j)$. This may look the same as the one for the first layer, but there is the big difference that the y_j are no longer independent. Their dependence does not, however, affect the evaluation of the mean of the outputs, which is

$$\overline{f}_i(\mathbf{s}) = \overline{b}_i + \sum_j \overline{B}_{ij} \overline{\varphi}(y_j). \tag{B.12}$$

The dependence between y_j arises from the fact that each s_i may potentially affect all of them. Hence, the variances of s_i would be taken into the account incorrectly if y_j were assumed independent.

Two possibly interfering paths can be seen in Figure B.1. Let us assume that the net weight of the left path is 1 and the weight of the right path -1, so that the two paths cancel each other out. If, however, the outputs of the hidden layer are incorrectly assumed to be independent, the estimated variance of the output will be greater than zero. The same effect can also happen the other way round when constructive inference of the two paths leads to underestimated output variance.

The effects of different components of the inputs on the outputs can be measured using the Jacobian matrix $\partial \mathbf{f}(\mathbf{s})/\partial \mathbf{s}$ of the mapping $\mathbf{f}$ with elements $(\partial f_i(\mathbf{s})/\partial s_j)$. This leads to the approximation for the output variance

$$\begin{aligned} \widetilde{f}_i(\mathbf{s}) \approx & \sum_j \left(\frac{\partial f_i(\mathbf{s})}{\partial s_j} \right)^2 \widetilde{s}_j + \widetilde{b}_i + \\ & \sum_j \left[\overline{B}_{ij}^2 \widetilde{\varphi}^*(y_j) + \widetilde{B}_{ij} \left(\overline{\varphi}^2(y_j) + \widetilde{\varphi}(y_j) \right) \right] \end{aligned} \tag{B.13}$$

where $\widetilde{\varphi}^*(y_j)$ denotes the posterior variance of $\varphi(y_j)$ without the contribution of the input variance. It can be computed as

$$\widetilde{y}_i^* = \widetilde{a}_i + \sum_j \widetilde{A}_{ij} \left(\overline{s}_j^2 + \widetilde{s}_j \right) \tag{B.14}$$

$$\widetilde{\varphi}^*(y_i) \approx \left[\varphi'(\overline{y}_i) \right]^2 \widetilde{y}_i^*. \tag{B.15}$$

The needed partial derivatives can be evaluated efficiently at the mean of the inputs with the chain rule

$$\frac{\partial f_i(\mathbf{s})}{\partial s_j} = \sum_k \frac{\partial f_i(\mathbf{s})}{\partial \varphi_k} \frac{\partial \varphi_k}{\partial y_k} \frac{\partial y_k}{\partial s_j} = \sum_k \overline{B}_{ik} \varphi'(\overline{y}_k) \overline{A}_{kj}. \tag{B.16}$$

Bibliography

[1] D. K. Arrowsmith and C. M. Place. *An Introduction to Dynamical Systems*. Cambridge University Press, Cambridge, 1990.

[2] David Barber and Christopher M. Bishop. Ensemble learning for multilayer networks. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems 10, NIPS*97*, pages 395–401, Denver, Colorado, USA, Dec. 1–6, 1997, 1998. The MIT Press.

[3] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[4] J. M. Bernardo. Psi (digamma) function. *Applied Statistics*, 25(3):315–317, 1976.

[5] Christopher Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[6] Christopher Bishop. Latent variable models. In Jordan [29], pages 371–403.

[7] Thomas Briegel and Volker Tresp. Fisher scoring and a mixture of modes approach for approximate inference and learning in nonlinear state space models. In Michael S. Kearns, Sara A. Solla, and David A. Cohn, editors, *Advances in Neural Information Processing Systems 11, NIPS*98*, pages 403–409, Denver, Colorado, USA, Nov. 30–Dec. 5, 1998, 1999. The MIT Press.

[8] Martin Casdagli. Nonlinear prediction of chaotic time series. *Physica D*, 35(3):335–356, 1989.

[9] Martin Casdagli, Stephen Eubank, J. Doyne Farmer, and John Gibson. State space reconstruction in the presence of noise. *Physica D*, 51(1–3):52–98, 1991.

[10] Vladimir Cherkassky and Filip Mulier. *Learning from Data: Concepts, Theory, and Methods*. John Wiley & Sons, New York, 1998.

[11] Y. J. Chung and C. K. Un. An MLP/HMM hybrid model using nonlinear predictors. *Speech Communication*, 19(4):307–316, 1996.

[12] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley series in telecommunications. John Wiley & Sons, New York, 1991.

[13] 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 (Methodological)*, 39(1):1–38, 1977.

[14] J. Doyne Farmer and John J. Sidorowich. Predicting chaotic time series. *Physical Review Letters*, 59(8):845–848, 1987.

[15] Ken-ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.

[16] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, Boca Raton, 1995.

[17] Zoubin Ghahramani. An introduction to hidden Markov models and Bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(1):9–42, 2001.

[18] Zoubin Ghahramani and Geoffrey E. Hinton. Variational learning for switching state–space models. *Neural Computation*, 12(4):831–864, 2000.

[19] Zoubin Ghahramani and Sam T. Roweis. Learning nonlinear dynamical systems using an EM algorithm. In Michael S. Kearns, Sara A. Solla, and David A. Cohn, editors, *Advances in Neural Information Processing Systems 11, NIPS*98*, pages 599–605, Denver, Colorado, USA, Nov. 30–Dec. 5, 1998, 1999. The MIT Press.

[20] Monson H. Hayes. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, New York, 1996.

[21] Simon Haykin. *Neural Networks – A Comprehensive Foundation, 2nd ed.* Prentice-Hall, Englewood Cliffs, 1998.

[22] Simon Haykin and Jose Principe. Making sense of a complex world. *IEEE Signal Processing Magazine*, 15(3):66–81, 1998.

[23] 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, USA, July 26-28, 1993.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Feature extraction through LOCOCODE. *Neural Computation*, 11(3):679–714, 1999.

[25] Antti Honkela and Juha Karhunen. An ensemble learning approach to nonlinear independent component analysis. In *Proceedings of the 15th European Conference on Circuit Theory and Design (ECCTD'01)*, Espoo, Finland, August 28–31, 2001. To appear.

[26] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[27] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis*. John Wiley & Sons, 2001. In press.

[28] O. L. R. Jacobs. *Introduction to Control Theory*. Oxford University Press, Oxford, second edition, 1993.

[29] Michael I. Jordan, editor. *Learning in Graphical Models*. The MIT Press, Cambridge, Massachusetts, 1999.

[30] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introdution to variational methods for graphical models. In Jordan [29], pages 105–161.

[31] Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME, Journal of Basic Engineering*, 82:35–45, 1960.

[32] Edward W. Kamen and Jonathan K. Su. *Introduction to Optimal Estimation*. Springer, London, 1999.

[33] Mikko Kurimo. *Using Self-Organizing Maps and Learning Vector Quantization for Mixture Density Hidden Markov Models*. PhD thesis, Helsinki University of Technology, Espoo, 1997. Published in Acta Polytechnica Scandinavica, Mathematics, Computing and Management in Engineering Series No. 87.

[34] Harri Lappalainen and Antti Honkela. Bayesian nonlinear independent component analysis by multi-layer perceptrons. In Mark Girolami, editor, *Advances in Independent Component Analysis*, pages 93–121. Springer, Berlin, 2000.

[35] Harri Lappalainen and James W. Miskin. Ensemble learning. In Mark Girolami, editor, *Advances in Independent Component Analysis*, pages 76–92. Springer, Berlin, 2000.

[36] Peter M. Lee. *Bayesian Statistics: An Introduction.* Arnold, London, second edition, 1997.

[37] David J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[38] 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.

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

[40] David J. C. MacKay. Choice of basis for Laplace approximation. *Machine Learning*, 33(1):77–86, 1998.

[41] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, New York, 1979.

[42] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 2. Academic Press, New York, 1982.

[43] Kevin Murphy. Switching Kalman filters. Technical report, Department of Computer Science, University of California Berkeley, 1998.

[44] Radford M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics.* Springer, New York, 1996.

[45] Radford M. Neal and Geoffrey E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan [29], pages 355–368.

[46] Jacob Palis, Jr and Welington de Melo. *Geometric Theory of Dynamical Systems.* Springer, New York, 1982.

[47] William D. Penny, Richard M. Everson, and Stephen J. Roberts. Hidden Markov independent component analysis. In Mark Girolami, editor, *Advances in Independent Component Analysis*, pages 3–22. Springer, Berlin, 2000.

[48] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[49] S. Neil Rasband. *Chaotic Dynamics of Nonlinear Systems.* John Wiley & Sons, New York, 1990.

[50] Sam T. Roweis and Zoubin Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11(2):305–345, 1999.

[51] Sam T. Roweis and Zoubin Ghahramani. An EM algorithm for indentification of nonlinear dynamical systems. Submitted for publication. Preprint available from `http://www.gatsby.ucl.ac.uk/~roweis/publications.html`, 2000.

[52] Walter Rudin. *Real and Complex Analysis.* McGraw-Hill, Singapore, third edition, 1987.

[53] Tim Sauer, James A. Yorke, and Martin Casdagli. Embedology. *Journal of Statistical Physics*, 65(3/4):579–616, 1991.

[54] Vesa Siivola. An adaptive method to achieve speaker independence in a speech recognition system. Master's thesis, Helsinki University of Technology, Espoo, 1999.

[55] Floris Takens. Detecting strange attractors in turbulence. In David Rand and Lai-Sang Young, editors, *Dynamical systems and turbulence, Warwick 1980*, volume 898 of *Lecture Notes in Mathematics*, pages 366–381. Springer, Berlin, 1981.

[56] Edmondo Trentin and Marco Gori. A survey of hybrid ANN/HMM models for automatic speech recognition. *Neurocomputing*, 37:91–126, 2001.

[57] Harri Valpola. *Bayesian Ensemble Learning for Nonlinear Factor Analysis.* PhD thesis, Helsinki University of Technology, Espoo, 2000. Published in Acta Polytechnica Scandinavica, Mathematics and Computing Series No. 108.

[58] Harri Valpola. Unsupervised learning of nonlinear dynamic state–space models. Technical Report A59, Helsinki University of Technology, Espoo, 2000.

[59] Harri Valpola, Xavier Giannakopoulos, Antti Honkela, and Juha Karhunen. Nonlinear independent component analysis using ensemble learning: Experiments and discussion. In Petteri Pajunen and Juha Karhunen, editors, *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, pages 351–356, Espoo, Finland, June 19–22, 2000.

[60] Harri Valpola and Juha Karhunen. An unsupervised ensemble learning method for nonlinear dynamic state–space models. A manuscript to be submitted to a journal, 2001.

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

[62] Hassler Whitney. Differentiable manifolds. *Annals of Mathematics*, 37(3):645–680, 1936.