HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Engineering
Physics and Mathematics

Tapani Raiko

Hierarchical Nonlinear Factor Analysis

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

Espoo, December 3, 2001

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

Teknillinen korkeakoulu — Diplomityön tiivistelmä
Teknillisen fysiikan ja matematiikan osasto

Tekijä:	Tapani Raiko
Osasto:	Teknillisen fysiikan ja matematiikan osasto
Pääaine:	Informaatiotekniikka
Sivuaine:	Matematiikka
Työn nimi:	Hierarkkinen epälineaarinen faktorianalyysi
Title in English:	Hierarchical Nonlinear Factor Analysis
Professuurin koodi ja nimi:	Tik-61 Informaatiotekniikka
Työn valvoja:	Prof. Juha Karhunen
Työn ohjaaja:	TkT Harri Valpola

Tiivistelmä:

Yleinen ongelmatyyppi koneoppimisen alalla on löytää joukosta näytevektoreita tilastollisia riippuvuuksia. Vektorin komponentit voivat olla mittauksia tehdasprosessista, kuvan pikselien väriarvoja tai muita reaalilukuja. Tässä työssä havainnoilla ei oleteta olevan erityistä järjestystä.

Faktorianalyysin tapaisissa menetelmissä sovitetaan havaintoihin niiden syntyä kuvaava generatiivinen malli, joka koostuu lähteistä (faktoreista) ja niiden kuvauksesta havaintoavaruuteen. Paljon käytetyissä menetelmissä, kuten pääkomponenttianalyysissä ja riippumattomien komponenttien analyysissä, on kuvaus rajoitettu lineaariseksi. Yleisessä epälineaarisessa tapauksessa muodostuvat laskennallinen vaativuus ja huono yleistyskyky helposti ongelmiksi. Bayesiläinen jakauma- eli ensemble-oppiminen on osoittautunut näihin varteenotettavaksi vastaukseksi ja lisäksi se mahdollistaa mallin rakenteen optimoinnin.

Tässä työssä esitellään ensemble-oppimiseen perustuva menetelmä, jossa generatiivinen malli rakennetaan yksinkertaisista paikallisesti opetettavista osista. Erityistä huomiota kiinnitetään epälineaarisuuteen. Paikallisuuden johdosta laskennallinen vaativuus on lineaarinen suhteessa rakenteen kokoon ja paloja eri tavoin yhdistämällä saadaan rakenteita, joilla voi muodostaa laajan valikoiman kuvauksia. Lähteet yhdistetään hierarkkisesti monikerros-perseptroni-verkon tavoin kuitenkin siten, että kerros kuvaa sekä edellisen kerroksen lähteiden varsinaisia arvoja, että niiden vaihtelua. Simulaatiot keinotekoisilla kuvapaloilla osoittavat että algoritmi löytää alkuperäisen kaltaisen generatiivisen mallin. Simulaatiot luonnollisilla kuvilla tuottivat mielenkiintoisia havaintoja ja kehitysideoita menetelmälle.

Sivumäärä: 89 — Avainsanat: epälineaarisuus, neuraalilaskenta, ensemble-oppiminen, faktorianalyysi, harvakoodaus

Täytetään osastolla

Hyväksytty: — Kirjasto:

Helsinki University of Technology — Abstract of Master's thesis
Department of Engineering Physics and Mathematics

Author:	Tapani Raiko
Department:	Department of Engineering Physics and Mathematics
Major subject:	Computer and Information Science
Minor subject:	Mathematics
Title:	Hierarchical Nonlinear Factor Analysis
Title in Finnish:	Hierarkkinen epälineaarinen faktorianalyysi
Chair:	Tik-61 Computer and Information Science
Supervisor:	Prof. Juha Karhunen
Instructor:	Harri Valpola, D.Sc. (Tech.)

Abstract:

A common problem class in machine learning is to find statistical dependencies among a group of data vectors. The elements of the vector can be measurements of an industry process, colour values of image pixels or some other real numbers. In this thesis, the observations are not assumed to have a certain order.

Methods related to factor analysis fit a generative model to the data. The model consists of sources (factors) and a mapping from source space to the data space. The mapping is restricted to be linear in many commonly used methods like principal component analysis and independent component analysis. In the general nonlinear case, computational complexity and the ability to generalise are problematic. Bayesian ensemble-learning has proved to answer to these challenges and it provides means to optimise the model structure.

This thesis presents an ensemble-learning based method in which a generative model is built from simple blocks with local update rules for learning. Special attention is given to the nonlinearity. It results from the local computations that the computational complexity is linear w.r.t. the size of the structure and a rich combination of mappings can be constructed by combining the blocks in various ways. A model is built hierarchically to resemble the multi-layer perceptron network, but in such a way that a layer represents both the actual values and the variation of the sources on the previous layer. Simulations with artificial image patches show that the algorithm can find a generative model similar to the original one. Simulations with natural images provide interesting findings and suggest improvements for the method.

Number of pages: 89	Keywords:	nonlinearity, neural networks, ensemble learning, factor analysis, sparse coding
Department fills		
Approved:	Library code:	

Preface

This work has been done in the Laboratory of Computer and Information Science at Helsinki University of Technology and funded by the European Commission research project BLISS.

I wish to thank professor Juha Karhunen for supervision and for financial support. I thank Dr. Harri Valpola for essential guidance on the subject. I also wish to thank Antti Honkela and Jaakko Peltonen for creative discussions and technical advice on Matlab, LaTeX and Unix.

Finally I wish to thank Anna Hiironen and my family for their encouragement and support.

Otaniemi, December 3, 2001

Tapani Raiko

Contents

List of abbreviations

DCT	Discrete cosine transform
EM	Expectation maximisation
FA	Factor analysis
HNFA	Hierarchical nonlinear factor analysis
HNFA+VM	Hierarchical nonlinear factor analysis with variance modelling
ICA	Independent component analysis
KL	Kullback-Leibler (divergence)
MAP	Maximum a posteriori (solution)
ML	Maximum likelihood (solution)
MLP	Multi-layer perceptron (network)
NCA	Nonlinear component analysis
NFA	Nonlinear factor analysis
PCA	Principal component analysis
pdf	Probability density function
RBF	Radial basis function
SOM	Self-organising map
std	Standard deviation

List of symbols

$\mathbf{x}(t)$	The data vector at time t
$\mathbf{A}$	The mixing matrix
$\mathbf{a}_k$	The kth principal component
$\mathbf{a}^T$	The transpose of $\mathbf{a}$
$\mathbf{s}(t)$	The source or factor vector at time t
$\boldsymbol{X}$	The data matrix containing all the observed data vectors
$\mathcal{H}$	The prior beliefs
$\boldsymbol{\theta}$	The parameters including the sources of a model
$p(A\|B)$	Probability density of A with condition B
$p(\boldsymbol{\theta}\|\boldsymbol{X},\mathcal{H})$	Posterior density
$p(\boldsymbol{X}\|\boldsymbol{\theta},\mathcal{H})$	Likelihood density
$p(\boldsymbol{\theta}\|\mathcal{H})$	Prior density
$p(\boldsymbol{X}\|\mathcal{H})$	Evidence density
$q(\boldsymbol{\theta})$	The parametric approximation of the posterior
C_{KL}	The Kullback–Leibler divergence
$D(q(\boldsymbol{\theta})\|\|p(\boldsymbol{\theta}\|\boldsymbol{X},\mathcal{H}))$	The Kullback–Leibler divergence between $q(\boldsymbol{\theta})$ and $p(\boldsymbol{\theta}\|\boldsymbol{X},\mathcal{H})$
$\langle\cdot\rangle$	Expectation over the distribution $q(\boldsymbol{\theta})$
$\mathrm{Var}\,\{\cdot\}$	Variance over the distribution $q(\boldsymbol{\theta})$
C	The cost function $C = C_q + C_p$
m	A signal that gives a prior mean of a source
v	A signal that gives a prior variance of a source
$f(\cdot)$	A nonlinearity (in most cases $f(s) = \exp(-s^2)$)
$\overline{s}$	Mean of the Gaussian distribution of $q(s)$
$\widetilde{s}$	Variance of the Gaussian distribution of $q(s)$
$\mathrm{N}\left(s;\overline{s},\widetilde{s}\right)$	Gaussian distribution of s with mean $\overline{s}$ and variance $\widetilde{s}$
t	The time index of an observation or source vector
k	The dimension of an observation or source vector

$s_{i,k}(t)$	The kth source on the ith layer for the tth observation
$u_{i,k}(t)$	The variance source attached to $s_{i,k}(t)$
$\mathbf{A}_i$	The matrix used for the mapping from $(i+1)$th layer to the sources of the ith layer
$\mathbf{B}_i$	The matrix used for the mapping from $(i+1)$th layer to the variance sources of the ith layer
$\mathbf{V}$	The matrix containing the orthonormal eigenvectors of the covariance of the data matrix
$\mathbf{D}$	The diagonal matrix containing the eigenvalues of the covariance of the data matrix
$\mathbf{M}$	The matrix containing the model vectors used in vector quantisation initialisation
β	The scaling factor used in initialisation

Chapter 1

Introduction

A flexible scheme to learn to classify, to infer and to predict things simply by observing examples, would be invaluable. Visualisation techniques help humans to perceive large data sets at a glance, but there is still trouble in perceiving data with high intrinsic dimensionality. An automatic system is not limited to any strict number of dimensions. As the processing power and the amount of available raw data grows, the greatest cost of analysation is the amount of human intervention.

Analysing images is an effective way to study these models since the results are relatively easy to interpret. Images can be considered as high dimensional data by thinking that each pixel is a component of a data vector. One observation vector is thus an image patch. The human brain is specialised among many other things to interpreting visual observations. Fairly sophisticated computer vision methods for certain tasks like for face recognition [68] exist, but they are highly specialised and thus incapable of adapting to new situations. The capabilities of such a system are limited by the engineering work, which easily builds up to the system until it is an incomprehensible mass of finely tuned rules.

1.1 Aim of the Thesis

The aim of this thesis is to build a system that is able to model high-dimensional static data such as image patches. The model could then be used for noise reduction, for reconstruction of missing values and thus to supervised learning tasks [57], for predicting future observations or as a part of an autonomous intelligent system. These practical applications are briefly discussed in Chapter 9.

Generative models are promising approaches to unsupervised learning tasks. A factor-analysis like latent variable model called hierarchical nonlinear factor analysis with variance modelling (HNFA+VM) is used in this thesis. It belongs to generative models. As the structure and the complexity of the model is highly adjustable, there is a need for a cost function that can be used for learning the model structure and balancing between over- and under-fitting. As the dimensionality of the data is high, the computational complexity is also a very important issue.

Ensemble learning [25] provides such a cost function. It can be used so that the complexity scales linearly with respect to the size of the system. Ensemble learning and related variational methods have been successfully applied to various extensions of linear Gaussian factor analysis. The extensions have included mixtures-of-Gaussian distributions for source signals [2, 10, 49], nonlinear units [15, 50] and MLP networks to model nonlinear observation mappings [43] and nonlinear dynamics of the sources [63]. Ensemble learning has also been applied to large discrete models such as belief networks [51] and hidden Markov models [48].

In this thesis, a model is presented which is built from addition, multiplication and Gaussian variables possibly followed by a nonlinearity. The Gaussian variables also model the variance of other Gaussian variables allowing also the variance to have a hierarchical model. Related model structures have been proposed for instance in [40, 8, 56, 19, 30, 29] but with these methods it is difficult to learn the structure of the model or compare different model structures.

1.2 Problem Settings

Two different experimental settings are used in this thesis. First an extension to the artificial bars problem [12] is studied to demonstrate that the algorithm actually finds a model that is similar to the one used to generate the data. Small image patches are generated by randomly inserting horisontal and vertical bars and areas with increased noise level to an image.

The second experiment setting consists of analysing patches of natural grayscale images. In this case there is no right answer, but the resulting model can be compared to what biologists know about the human brain. Hyvärinen and Hoyer [30, 29, 31] applied independent component analysis (ICA) and its extensions to natural images. The basic ICA leads to emergence of simple cell properties and the extensions lead to emergence of both topography and invariances similar to complex cell properties.

Frey and Jojic estimated a mixture model for images with a fixed set of transformations [16]. Using 100 frames of head-and-shoulder video sequences of a person walking across a higly cluttered background and a fixed set of translational invariances, they could capture clusters that presented the person with different poses, while background was interpreted as noise.

1.3 Structure and Contributions of the Thesis

This thesis is organised as follows. Chapter 2 discusses previous work on extensions of factor analysis that can be used for unsupervised learning. Chapter 3 gives an overview of Bayesian ensemble learning which is the essential theoretical background. Building blocks and their usage with ensemble learning are described in Chapter 4. The model structure used in hierarchical nonlinear factor analysis with variance modelling (HNFA+VM) is built from these blocks in Chapter 5. Chapter 6 describes the algorithm that is used to let the model learn from the data.

Two sets of experiments were conducted using HNFA+VM. In Chapter 7, an artificial bars problem is analysed and in Chapter 8, the model is applied on natural image data. Finally, the benefits, restrictions and applications of the model and future work are discussed in Chapter 9.

The update rule for the Gaussian variable with nonlinearity has been developed by the author. The experiments as well as the code for preprocessing and parts of the learning procedure like initialisation, pruning, regeneration and rebooting, have beed developed by the author.

Chapter 2

Extensions of Factor Analysis

This chapter describes shortly the ideas behind factor analysis and how it has been extended. It also describes how the model in this thesis is related to other work in the field.

In factor analysis, regularities in the observations are assumed to be caused by hidden factors or sources. The problem setting in all the factor-analysis like models is to find those factors and the mapping to the data. These models are generative which means that they describe a random process which is thought to have generated the data. Generative models have some useful properties. The learning criterion is easy to formulate and thus includes no heuristics. The models can be combined and their performances can be compared. They also offer a straightforward way to handle missing values.

The data is in this case a collection of real-valued vectors $\mathbf{x}(t)$, where $t \in \{1, 2, \ldots, T\}$ is the time index for the observation. In static analysis, the dependencies between consequent time indices are neglected and the data could be permutated, still giving the same result as opposed to dynamical analysis. The mean of the data is assumed to be a zero vector to simplify the equations.

2.1 Linear Models

2.1.1 Factor Analysis

The basic approach is simply called factor analysis [22, 38] (FA). The data $\mathbf{x}(t)$ is thought to have been generated from factors $\mathbf{s}(t)$ through a linear mapping

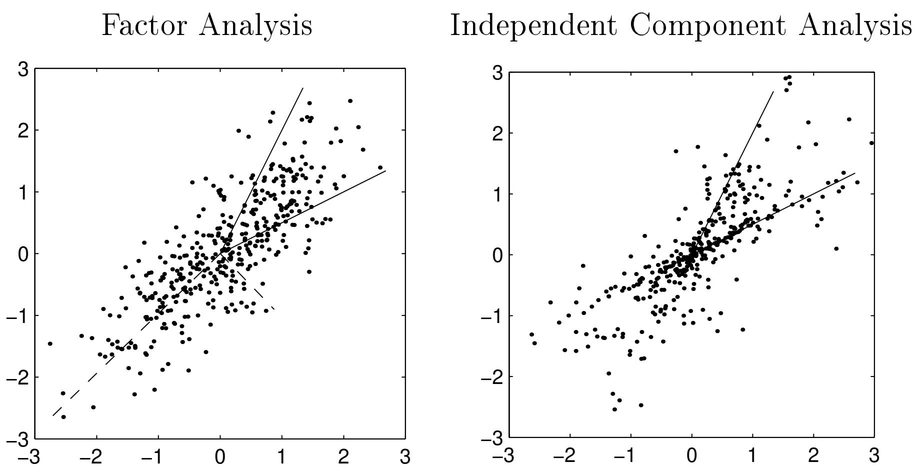


Figure 2.1: Left: Factor Analysis. The image has been generated by mixing two Gaussian factors with a matrix visualised with the solid lines. In factor analysis, the rotation is not fixed, but principal component analysis selects the orientation with orthogonal axis shown with the dash lines. Right: When the factors have non-Gaussian distributions the rotation is fixed and can be found using independent component analysis.

$\mathbf{A}$ using the formula

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t) + \mathbf{n}(t), \tag{2.1}$$

where $\mathbf{n}(t)$ is noise or reconstruction error vector. The vectors $\mathbf{A}\mathbf{s}(t)$ are called the reconstructions. Typically the dimensionality of the factors is smaller than that of the data. Factors and noise are assumed to have a Gaussian distribution. Figure 2.1 shows a two dimensional example. Factor analysis can be seen as trying to fit a Gaussian distribution to the data points.

Equation 2.1 does not fix the matrix $\mathbf{A}$, since there is a group of rotations that yield identical observation distributions. Several criteria have been suggested for determining the rotation. One is called the parsimony, which roughly means that most of the values in $\mathbf{A}$ are close to zero. Others include independency of the factors and discarding the assumption of Gaussian distribution of them and instead maximising the nongaussianity. These have led to independent component analysis [31, 35] (ICA) that is described below.

2.1.2 Principal Component Analysis

Principal component analysis (PCA)[32, 38] also known as the Hotelling transform or the Karhunen-Loève transform is a widely used method for finding the most important directions in the data in the mean-square sense. It is the solution of the FA problem with minimum mean square error and an orthogonal weight matrix.

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

$$\mathbf{a}_1 = \arg \max_{||\mathbf{a}||=1} \sum_{t=1}^{T} (\mathbf{a}^T \mathbf{x}(t))^2. \tag{2.2}$$

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

$$\mathbf{a}_k = \arg \max_{||\mathbf{a}||=1} \sum_{t=1}^{T} \left[\mathbf{a}^T \left(\mathbf{x}(t) - \sum_{i=1}^{k-1} \mathbf{a}_i \mathbf{a}_i^T \mathbf{x}(t) \right) \right]^2. \tag{2.3}$$

In practice, the principal components are found by calculating the eigenvectors of the covariance matrix $\mathbf{C}$ of the data

$$\mathbf{C} = E \left\{ \mathbf{x}(t)\mathbf{x}(t)^T \right\} \tag{2.4}$$

The eigenvalues are positive and they correspond to the variances of the projections of data on the eigenvectors.

Principal components can be found in various fields of science. For example, there is an analogy to the physics. If three dimensional data points are considered to be the mass points of a rigid body, the eigenvalues correspond to the principal moments of inertia and the principal components to the principal axes of the body.

2.1.3 Independent Component Analysis

The mixing model of independent component analysis (ICA) is similar to that of the FA, but in the basic case without the noise term. The data has been generated from components $\mathbf{s}(t)$ through a square mixing matrix $\mathbf{A}$ by

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t). \tag{2.5}$$

The distribution of the components are assumed to be non-Gaussian contrary to FA. Figure 2.1 shows an example with super-Gaussian components.

The number of components is typically the same as the number of observation and the observations can be thus reconstructed perfectly. Such an $\mathbf{A}$ is searched for that the components $\mathbf{s}(t) = \mathbf{A}^{-1}\mathbf{x}(t)$ would be 'as independent as possible'. True independence is defined as

$$p(s_1, s_2, \ldots, s_n) = p_1(s_1)p_2(s_2)\ldots p_n(s_n), \tag{2.6}$$

which means that the density is factorised to a product of the marginal densities of the components. In practice, the independence can be maximised e.g. by maximising non-Gaussianity of the components or minimising mutual information [31].

ICA has many forms [31]. It can be approached from different starting points. In some extensions the number of independent components can exceed the number of dimensions of the observations making the basis overcomplete [46, 31]. The noise term can be taken into the model. ICA can be viewed as a generative model when the one dimensional distributions for the components are modelled with for example mixtures-of-Gaussians [2, 10, 49].

Experiments have shown that some components found by ICA tend to be active, i.e. nonzero simultaneously in most natural data. Taking this into account in the model has led to topographic ICA and independent subspace analysis [30, 29, 31]. In topographic ICA, the components are organised in a grid. The model states that the energies $s_i(t)^2$ and $s_j(t)^2$ of components i and j are positively correlated, if the components are close to each other in the grid. In independent subspaces, the components are grouped to feature subspaces such that inside the group the probability distribution is spherically symmetric and the energies are correlated. The groups are considered mutually independent.

2.2 Nonlinear Models

Nonlinear extensions of factor analysis can be divided into at least three categories: 1) Using multiple clusters that each resemble FA; 2) Mapping the data nonlinearly to a higher dimensional feature space and then performing linear computations there; 3) Finding the nonlinear shape of the data distribution. The model used in this thesis fits into the third category.

Examples from each category will be described. From the first category, a mixture model is considered. The self-organising map fits somewhere between the first and the third category. Nonlinear Hebbian learning [54, 53] and nonlinear component analysis (NCA) or kernel-PCA are examples of the second category. Examples from the third category include nonlinear factor analysis and principal curves [23]. Most of the previous work in nonlinear supervised learning tasks can be applied to nonlinear factor analysis and therefore they are considered for background.

2.2.1 Mixtures of Linear Models

The first approach to extend factor analysis to nonlinear manifolds is to use a mixture model

$$\mathbf{x}(t) = \mathbf{A}_k\mathbf{s}(t) + \mathbf{a}_k + \mathbf{n}(t), \tag{2.7}$$

where k is a discrete random variable that selects the kernel or the mixing matrix and the corresponding bias. The bias term $\mathbf{a}_k$ is needed here even if the data is centered i.e. it has zero mean. The mixture model can be seen as trying to cover the data points precisely with a distribution containing several Gaussians.

Bishop et al. [7] used this kind of a model with Bayesian inference to image modelling tasks. There was considerable enhancement in the performance when compared to linear methods.

A mixture model is often inadequate with high dimensional data like images. To represent shapes and colours of different object in an image, one would need a componentwise representation of the parameters of the objects. But to represent the multiple objects, one would need several representations active at one time, which is not allowed.

It is possible to restrict the model in (2.7) by using only diagonal mixing matrices $\mathbf{A}_k$, leaving out the whole term $\mathbf{A}_k\mathbf{s}(t)$ or leaving out the noise term $\mathbf{n}(t)$. Restrictions simplify the model and thus one can use a greater number of kernels with the same total complexity. Restricted models come close to vector quantisation [1]. Mixture models have also been extended to mixture-of-ICA models [36, 45].

2.2.2 Nonlinear Component Analysis

The basic idea of nonlinear component analysis (NCA) or kernel PCA [60] is to replace the covariance matrix in equation (2.4) with

$$\mathbf{C} = E\left\{\Phi(\mathbf{x}(t))\Phi(\mathbf{x}(t))^T\right\}, \tag{2.8}$$

where Φ is a fixed nonlinear mapping to a feature space which has a larger dimensionality than the data space. The principal components are then computed in the feature space.

The mapping Φ is typically not a surjection, i.e. onto and therefore the reconstruction of a data vector given the extracted components can be problematic. Linear principal components can be visualised easily since they correspond to vectors or directions in the data space, whereas the kernel-based principal components do not have a counterpart in the data space. Therefore NCA is not viewed as a generative model.

Schölkopf et al. [60] have developed an efficient algorithm for NCA where the dimensionality of the feature space can be very large. They also developed a method for iteratively finding a reconstruction in the data space for a point in the feature space. Their experiments suggest that compared to PCA, NCA extracts features which are more useful for classification purposes. The same approach can be used to construct e.g. nonlinear ICA.

2.2.3 Self-Organising Map

The self-organising map (SOM) [39], also known as the Kohonen map, lies in a sense between vector quantisation and nonlinear factor analysis. It operates with map units which are comparable to kernels of mixture models. The map units are organised into a typically two-dimensional grid, where the model vectors of neighbours in the grid are neighbours in the data space.

The actual learning of SOM is done basically as follows. A data sample $\mathbf{x}(t)$ is compared to the model vectors $\mathbf{m}_i$ and the winner index $c(\mathbf{x}(t))$ is selected with

$$c = \arg\min_i |\mathbf{x}(t) - \mathbf{m}_i| \tag{2.9}$$

and the model vectors are adjusted towards the data vector

$$\mathbf{m}_{i,new} = \mathbf{m}_i + h(c(\mathbf{x}(t)), i)(\mathbf{x}(t) - \mathbf{m}_i), \tag{2.10}$$

where the h is called the neighbourhood function. It is a decreasing function of the distance between the the ith and cth model on the map grid.

If the map grid is just two dimensional, it is useful for visualisation. When the dimensionality gets larger, however, the number of map units grows exponentially and therefore the model is not well suited for tasks with high intrinsic dimensionality. There are plenty of modifications [39] to the basic SOM. The so called generative topographic map [6] is a generative model which is closely related to the SOM.

2.2.4 Supervised Learning Tasks

Nonlinear models have long been used for supervised learning tasks. The task in supervised learning is to learn a mapping $\mathbf{f}$ from observation pairs $\{\mathbf{x}(t), \mathbf{d}(t)\}$. The model is typically written as

$$\begin{aligned} \mathbf{y}(t) &= \mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta}) & (2.11) \\ \mathbf{e}(t) &= \mathbf{d}(t) - \mathbf{y}(t) & (2.12) \end{aligned}$$

where $\mathbf{y}(t)$ is the output of the network $\mathbf{f}$ and $\mathbf{e}(t)$ is the error signal or the difference between the desired response $\mathbf{d}(t)$ and the output $\mathbf{y}(t)$.

The mapping $\mathbf{f}$ has a fixed structure and parameters $\boldsymbol{\theta}$ control the actual shape of the mapping. The essential difference to factor-analysis-like models is that here both the inputs and outputs are observed. The only hidden variables that have to be learned are the parameters $\boldsymbol{\theta}$ that control the shape of the mapping $\mathbf{f}$.

One of the most common structures for $\mathbf{f}$ is the multilayer perceptron (MLP) network [5, 24] with one hidden layer defined by

$$\mathbf{y}(t) = \mathbf{A}_1\mathbf{g}(\mathbf{A}_2\mathbf{x}(t) + \mathbf{a}_2) + \mathbf{a}_1, \qquad (2.13)$$

where $\mathbf{g}(\cdot)$ is a nonlinear activation function like the hyperbolic tangent applied to each component of the vector separately. It is an universal approximator [26, 17], which means that given enough hidden units, it can approximate any measurable function to any desired degree of accuracy. The learning can be done by adjusting the parameters $\mathbf{A}$ and $\mathbf{a}$ so that the error signal $\mathbf{e}(t)$ gets close to zero. There are plenty of other structures and methods [24].

2.2.5 Nonlinear Factor Analysis

In nonlinear factor analysis (NFA), the mixing matrix used in FA is replaced by a general mixing model

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t). \qquad (2.14)$$

For the mapping $\mathbf{f}$, one can use the same collection of structures as in the supervised tasks.

The assumption that the factors have a Gaussian distribution like in the basic FA is not restrictive anymore since the nonlinear mapping can compensate for it. The complexity is typically polynomial w.r.t. the number of factors which means that data with intrinsic dimensionality of the order of ten can be handled. It should be noted, however, that if the data is clustered, the corresponding mapping $\mathbf{f}$ would be very highly nonlinear and difficult to learn. Other methods are better suited for clustering.

If the mapping $\mathbf{f}$ is modelled with for example an MLP network with one hidden layer, an analogy to nonlinear component analysis (NCA) can be found. There is a nonlinear mapping between the data space and the higher dimensional feature space or hidden layer. The mapping between the underlying factors and the feature space or hidden layer is linear. The main difference is that in NCA, the mapping to feature space is predefined, but in NFA, the mapping from hidden layer to data is learned, too. The feature space in NFA is typically of higher dimensionality than the hidden layer in NFA.

The solutions of NFA are highly nonunique [31]. When compared to supervised learning where only the mapping $\mathbf{f}$ can be adjusted, one finds out that in NFA, many mappings are reasonably good, since the factors $\mathbf{s}(t)$ can be adjusted accordingly. NCA is comparable to NFA with part of the mapping fixed. There is a downside to the freedom of NFA: simple learning criteria lead to bad overfitting as will be discussed in Section 2.4. The problem can be overcome with e.g. Bayesian learning.

Valpola et al. [43, 64] used an MLP network to model the mapping of NFA and ensemble learning for finding the parameters. Experiments showed that compared to the linear FA, a smaller number of factors was required to acquire the same level of accuracy in the reconstructions. Experiments with a modification by the present author and Valpola [57] showed that NFA performed better at reconstructing missing values in observations than FA.

2.2.6 Nonlinear Dynamical Factor Analysis

The NFA model can be extended to model the nonlinear dynamics of the factors [20, 65, 63, 64]

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{s}(t)) + \mathbf{n}(t) \tag{2.15}$$

$$\mathbf{s}(t) = \mathbf{g}(\mathbf{s}(t-1)) + \mathbf{m}(t), \tag{2.16}$$

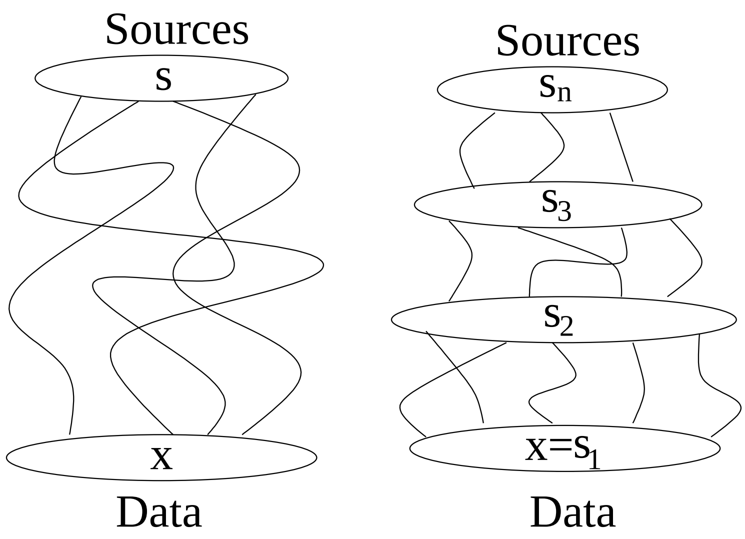


Figure 2.2: If the mapping from the sources to the data is strongly nonlinear, it might be much easier to learn it in parts as in the figure on the right.

where the $\mathbf{m}(t)$ is called process noise or innovation process. The mapping $\mathbf{g}$ modelling the dynamics is handled somewhat similarly as the mapping $\mathbf{f}$. Experiments [65, 63] have given good results in prediction of artificial time series compared to traditional time series methods.

2.2.7 Hierarchical Nonlinear Factor Analysis

In some of the described models the data is thought to have been generated from original factors or source signals through a linear or nonlinear mapping. The mapping and the sources can be adjusted to match the data better by propagating the reconstruction error of the data upwards in Figure 2.2. If the nonlinearity is strong, but the mapping is not yet very meaningful, it is hard to determine how the adjustments should be made.

In hierarchical nonlinear factor analysis (HNFA), the hidden units or calculating units of an MLP-like network in NFA are replaced by latent variables. Even if the mapping between two adjacent layers is only slightly nonlinear, the total mapping through all the layers can be strongly nonlinear. The learning or the adjustments can be done layer by layer. One part can learn even if the total mapping is not yet very meaningful.

Hierarchical models for parameters are widely used in modern Bayesian data analysis [18, 59]. That is, observations are modelled conditionally on some parameters which themselves are modelled by hyperparameters. In HNFA, however, the hierarchy applies also to the time dependent sources and not only to the parameters.

HNFA belongs to Bayesian networks [33]. In Bayesian networks, the variables are connected as a directed acyclic graph. Some of the variables are observed and others are latent or hidden. Variables can be continuous valued or discrete.

Neal developed the logistic belief net [51], which is like HNFA with binary variables. He replaced symmetric connections of Boltzmann machine with directed connections that form an acyclic graph. After that, the probabilistic calculations become easy.

Frey and Hinton [15] constructed a nonlinear Gaussian belief network for tasks such as stereo vision and speech recognition. Gaussian latent variables were passed through linear, binary, rectified and sigmoidal functions to get nonlinear units. Maximum likelihood and Gibbs sampling were compared as learning methods.

Murphy [50] used a variational approximation to the logistic function to perform approximate inference in Bayesian networks containing discrete nodes with continuous parents. The experiments showed that the variational approximation is much faster than sampling, but comparable in accuracy.

2.3 Sparse Coding

In many of the described models, the observations are reconstructed as a weighted sum of model vectors. In vector quantisation [1], only one of the model vectors is active at a time, which means that all but one of the weights are zero. This is called *local coding*. There has to be lots of model vectors to get a reasonable accuracy. In basic factor analysis, the factors have a Gaussian distribution and therefore most of them can be active or nonzero at a time. This is called *global coding*. *Sparse coding* fits in between these two extremes; Each observation is reconstructed using just a few active units out of a larger collection. Biological evidence suggests [14] that the human brain uses sparse coding. It is said to have benefits like good representational capacity, fast learning, good generalisation and tolerance to faults and damage.

Sparsity implies that the distribution of a source has a high peak, but the variation can still be broad. The peak corresponds to the inactivity and therefore

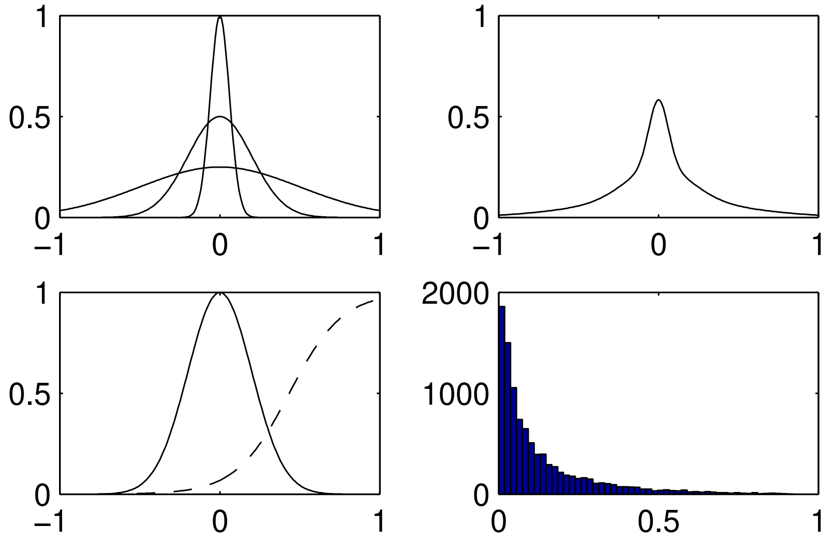


Figure 2.3: Two ways to obtain a supergaussian distribution from a Gaussian one. Top: The mean of three Gaussians with different variances is taken. Bottom: a Gaussian is fed through a nonlinearity marked with the dashed curve. The resulting distributions are shown on the right hand side.

it is typically at zero. Kurtosis is one measure of peakedness. It is defined as

$$\mathrm{kurt}(s) = E\{s^4\} - 3\left[E\{s^2\}\right]^2, \qquad (2.17)$$

if s is assumed to have zero mean. For Gaussian distributions, the kurtosis is zero and for peaked and heavy tailed, it is positive. ICA can be carried out by maximising the absolute values of the kurtosis which means that it can promote sparsity.

Most of the probability density functions in this thesis are Gaussian. However, by varying the variance or using a nonlinearity with a flat part as in Figure 2.3, a peaked distribution can be obtained. Beale showed [4] that by varying the variance of a symmetrical distribution, the kurtosis always increases. Because of the biological and experimental motivation to use sparse coding, it is made sure in Chapter 5 that the model is rich enough to use it.

2.4 Learning Criteria

Solving problems with complex models cannot be done analytically. Instead, one can formulate a learning criterion typically in a form of a cost function.

The learning can done by adjusting parameters iteratively such, that the cost function is minimised. Once the criterion is determined, one can use methods of optimisation theory, such as gradient descent.

The first candidate for a cost function could be the mean square reconstruction error

$$C_{ms} = \frac{1}{T} \sum_{t=1}^{T} \left[\mathbf{x}(t) - \mathbf{f}(\mathbf{s}(t)) \right]^2 . \tag{2.18}$$

It is easy to see [5] that it is equivalent to maximising the likelihood of the data $p(\mathbf{x}|\mathbf{s}, \mathbf{f}(\cdot))$ assuming a Gaussian noise with equal variance on each component of $\mathbf{x}(t)$. A logarithm of the reconstruction error is linearly dependent on the likelihood of the noise term. This approach works fine with supervised learning tasks, when the number of free parameters is relatively small.

In the basic ICA, the dimensionality of the model is the same as the dimensionality of the data. The reconstructions are perfect and the learning criterion cannot be based on reconstruction error. Instead, one can maximise the non-gaussianity of the components. Also, when the curvature is allowed to be high enough in a nonlinear model, the data can be reconstructed perfectly. That does not guarantee that the model would be meaningful or generalise to new samples. The problem is called overfitting and one needs a better criterion for learning something meaningful.

With simple criteria, the amount of data required for avoiding overfitting is directly proportional to the number of free parameters [24]. In supervised learning, increasing the number of data points will always overcome the problem. In unsupervised learning, however, the number of unknown variables grows with the number of data points, since the factors corresponding to each observation are also unknown. This would suggest that the simple methods work only, when the number of factors is small enough compared to the dimensionality of the data. Even that is not true if one tries to estimate the variance of the noise. This will be demonstrated in Section 3.3.2. Bayesian inference solves the problem of overfitting and it is addressed in Chapter 3.

Chapter 3

Bayesian Inference and Ensemble Learning

This chapter gives an overview of ensemble learning with emphasis on solutions yielding linear computational complexity. More comprehensive introductions to ensemble learning can be found for instance in [61, 34, 44].

In generative models, inference can be done by estimating how likely a set of parameter values could have produced the observed data. Basically one wants to find a representative of the probability mass in the parameter space, but there are different methods for doing it in practice.

3.1 Bayesian Probability Theory

The definition of probabities using frequencies will run into problems in some cases. What is the probability of six when a player throws a dice? One out of six. But what if the dice was biased? That is a good hypothesis and it should be tested. After trying a hundred throws the player is somewhat confident that the dice is biased. What is the probability of another six? How does it change exactly?

Another example is that a mathematician has a hypothesis, which he thinks is probably true. After writing the first half of the proof and sketching the other half he is somewhat more certain that the proof exists and the hypothesis is true. It is clear that in this case the probability is a subjective degree of belief. The frequency of a certain mathematical hypothesis to be true or not is quite absurd as a concept in this example.

In contrast to the traditional definition of probability using relative frequencies, the Bayesian probability theory interprets probability as a degree of belief. This offers a way to put all the hypotheses to prior beliefs $\mathcal{H}$. The data or the observations $\boldsymbol{X}$ are in the player example the dice throws and the parameters $\boldsymbol{\theta}$ are the studied properties of the dice.

Cox has shown [11] that from very general requirements of consistency and compatibility with common sense, the basic laws concerning beliefs are equivalent to

$$p(A, B \mid C) = p(A \mid C)p(B \mid A, C) \tag{3.1}$$

$$p(A \mid B) + p(\bar{A} \mid B) = 1, \tag{3.2}$$

from which the two basic rules of Bayesian inference, Bayes rule and the marginalisation principle can be derived.

3.1.1 Bayes Rule

The Bayes rule

$$p(\boldsymbol{\theta} \mid \boldsymbol{X}, \mathcal{H}) = \frac{p(\boldsymbol{X} \mid \boldsymbol{\theta}, \mathcal{H})p(\boldsymbol{\theta} \mid \mathcal{H})}{p(\boldsymbol{X} \mid \mathcal{H})} \tag{3.3}$$

indicates how observations change the beliefs. It is named after an English reverend Thomas Bayes, who lived in the 18th century. $\mathcal{H}$ marks the prior beliefs, $\boldsymbol{X}$ is the observed data and $\boldsymbol{\theta}$ is the parameter vector to be inferred. The term $p(\boldsymbol{\theta} \mid \mathcal{H})$, the probability of the parameters given only the prior beliefs or the probability prior to the observations is called the *prior probability*. The term $p(\boldsymbol{\theta} \mid \boldsymbol{X}, \mathcal{H})$, the probability of the parameters given both the observations and the prior beliefs is called the *posterior probability* of the parameters. The Bayes rule tells how the prior probability is replaced by the posterior after getting the extra information i.e. the observed data. The term $p(\boldsymbol{X} \mid \boldsymbol{\theta}, \mathcal{H})$ is called the *likelihood* of the data and the term $p(\boldsymbol{X} \mid \mathcal{H})$ the *evidence* of the data. Note that here $p(\cdot)$ stands for a probability density function (pdf) over a vector space.

When inferring the parameter values $\boldsymbol{\theta}$, the evidence term is constant and the learning rule (3.3) simplifies to

$$p(\boldsymbol{\theta} \mid \boldsymbol{X}, \mathcal{H}) \propto p(\boldsymbol{X} \mid \boldsymbol{\theta}, \mathcal{H})p(\boldsymbol{\theta} \mid \mathcal{H}) \tag{3.4}$$

Bayes rule tells a learning system (the player) how to update its beliefs after observing $\boldsymbol{X}$. Now there is no room for 'what if', since the posterior contains everything that can be inferred from the observations. Note that the learning cannot start from void - some prior beliefs are always necessary as will be discussed in Section 3.3.3.

3.1.2 Marginalisation Principle

The marginalisation principle specifies how a learning system can predict or generalise. The probability of observing A with prior knowledge of B is

$$p(A \mid B) = \int p(A \mid \boldsymbol{\theta}, B) p(\boldsymbol{\theta} \mid B) d\boldsymbol{\theta}. \tag{3.5}$$

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

Using the principle, the evidence term can be written as

$$p(\boldsymbol{X} \mid \mathcal{H}) = \int p(\boldsymbol{X} \mid \boldsymbol{\theta}, \mathcal{H}) p(\boldsymbol{\theta} \mid \mathcal{H}) d\boldsymbol{\theta}. \tag{3.6}$$

This emphasises the role of the evidence term as a normalisation coefficient. It is an integral over the numerator of the Bayes rule (3.3).

3.1.3 Approximations

In practice, exact treatment of the posterior probability density of the parameters is infeasible except for very simple models. Therefore, some suitable approximation method must be used. There are at least three options: 1) a point estimate; 2) sampling; 3) parametric approximation.

The result of inferring the parameter values is called the solution. The Bayesian solution is the whole posterior pdf. In point estimates the solution is a single point. The maximum likelihood (ML) solution for the parameters $\boldsymbol{\theta}$ is the point in which the likelihood $p(\boldsymbol{X} \mid \boldsymbol{\theta}, \mathcal{H})$ is highest. The maximum a posteriori (MAP) solution is the one with highest posterior pdf $p(\boldsymbol{\theta} \mid \boldsymbol{X}, \mathcal{H})$. Point estimates are easiest to calculate but they fail in some situations as will be seen in Section 3.3.2.

It is possible [18] to construct a Markov chain that will draw points $\boldsymbol{\theta}_k$ from the posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{X}, \mathcal{H})$. Instead of integrating over the posterior distribution in (3.5), one can sum over the sequence $\boldsymbol{\theta}_k$. This sampling approach has originated from the Metropolis algorithm in the statistical physics and developed into e.g. Gibbs sampling and hybrid Monte Carlo method. Generally they are called Markov chain Monte Carlo methods. The length of the sequence for the approximation to be adequate can get too large to be used in practice when the problem is hard and has a large number of parameters.

Ensemble learning is a compromise between the Bayesian solution and the point estimates. It is used in this thesis and therefore the whole Section 3.2 is dedicated to it.

3.2 Ensemble Learning

Ensemble learning [25, 3, 44, 52] is one type of variational learning. The basic idea in ensemble learning is to minimise the misfit between the exact posterior pdf $p(\boldsymbol{\theta} \mid \boldsymbol{X})$ and its parametric approximation $q(\boldsymbol{\theta})$. The misfit is measured with the Kullback-Leibler (KL) divergence

$$\begin{aligned} C_{\mathrm{KL}} = D(q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta} \mid \boldsymbol{X}, \mathcal{H})) &= \left\langle \ln \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta} \mid \boldsymbol{X}, \mathcal{H})} \right\rangle \qquad (3.7) \\ &= \int q(\boldsymbol{\theta}) \ln \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta} \mid \boldsymbol{X}, \mathcal{H})} d\boldsymbol{\theta} \geq 0, \end{aligned}$$

where the operator $\langle \cdot \rangle$ denotes an expectation over the distribution $q(\boldsymbol{\theta})$. KL divergence is hard to evaluate and therefore the cost function C that is actually used is

$$C = \left\langle \ln \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{X}, \boldsymbol{\theta} \mid \mathcal{H})} \right\rangle = C_{\mathrm{KL}} - \ln p(\mathbf{X} \mid \mathcal{H}). \qquad (3.8)$$

The terms C and C_{KL} differ by the term $\ln p(\mathbf{X} \mid \mathcal{H})$, the logarithm of the evidence. While optimising the distribution $q(\boldsymbol{\theta})$ for a single model $\mathcal{H}$, the evidence is constant and can be ignored.

The cost function can be divided to a sum $C = C_q + C_p$, where

$$\begin{aligned} C_q &= \langle \ln q(\boldsymbol{\theta}) \rangle \qquad (3.9) \\ C_p &= -\langle \ln p(\boldsymbol{X}, \boldsymbol{\theta} \mid \mathcal{H}) \rangle . \qquad (3.10) \end{aligned}$$

Density estimates of continuous valued latent variables offer a great advantage over point estimates in being robust against over-fitting and providing a cost function suitable for learning model structures. With ensemble learning the density estimates can be almost as efficient as point estimates. Roberts [58] compared Laplace approximation, sample based and ensemble learning with ICA problem on music data. The sources were well recovered using ensemble learning and the approach was considerably faster than the other methods.

3.2.1 Factorial Approximation

One can choose a posterior approximation $q(\boldsymbol{\theta})$ that can be written as a product of independent densities

$$q(\boldsymbol{\theta}) = q(\theta_1)q(\theta_2)\dots q(\theta_N), \tag{3.11}$$

where N is the number of parameters. This will simplify C_q in (3.9) to a sum of simple terms. Factorial approximation allows linear computational complexity.

The factorial approximation allows building of efficient algorithms, but it might not work well with all kinds of model structures. It does work well with models in which the posterior dependencies are not too strong, like in the sparse coding. It seems that almost maximally factorial $q(\boldsymbol{\theta})$ suffices for latent variable models, since the rotational and other invariances can be used by choosing a solution where the factorial approximation is most accurate. A good model structure seems to be more important than a good approximation of the posterior probability of the model.

Miskin and MacKay [49] used ensemble learning for blind source separation. They compared two approximations of the posterior: The first was an M-dimensional Gaussian with full covariance matrix, which resulted in a memory requirement of the order M^2 and a time complexity of the order M^3. The second was the factorial approximation. They noticed that the factorial approximation is computationally more efficient and still gives a bound on the evidence and does not suffer from overfitting.

3.2.2 Hierarchical Models

In a hierarchical model, the data $\boldsymbol{X}$ depends only on some of the parameters. They are called the parameters of the first layer $\boldsymbol{\theta}_1$. Parameters of the first layer depend only on the second layer parameters $\boldsymbol{\theta}_2$ and so on.

The term $p(\boldsymbol{\theta}, \boldsymbol{X} \mid \mathcal{H})$ in equation (3.10) can be split into a product of simpler terms since dependencies over layers $p(\boldsymbol{\theta}_i|\boldsymbol{\theta}_{i+1}, \dots, \boldsymbol{\theta}_n) = p(\boldsymbol{\theta}_i|\boldsymbol{\theta}_{i+1})$ can be truncated

$$\begin{aligned} p(\boldsymbol{\theta}, \boldsymbol{X} \mid \mathcal{H}) &= p(\boldsymbol{X}|\boldsymbol{\theta}, \mathcal{H})p(\boldsymbol{\theta} \mid \mathcal{H}) \\ &= p(\boldsymbol{X}|\boldsymbol{\theta}_1, \mathcal{H})p(\boldsymbol{\theta}_1|\boldsymbol{\theta}_2, \mathcal{H}) \cdots p(\boldsymbol{\theta}_{n-1}|\boldsymbol{\theta}_n, \mathcal{H})p(\boldsymbol{\theta}_n \mid \mathcal{H}), \end{aligned} \tag{3.12}$$

where n is the number of layers.

This means that also the expectation in (3.10) and thus the whole cost function C in (3.8) become sums of simple terms.

3.2.3 Connection to coding

There is a close connection between coding and probabilistic framework. The optimal code length of $\boldsymbol{X}$ is $L(\boldsymbol{X}) = -\log_2 P(\boldsymbol{X})$ bits, where $P(\boldsymbol{X})$ is the probability of observing $\boldsymbol{X}$. In order to be able to encode the data compactly one has to find a good model for it. A sender and a receiver have agreed on a model structure $\mathcal{H}$ and the message will have two parts: the parameters $\boldsymbol{\theta}$ and the data $\boldsymbol{X}$. It can be shown[25] that $L(\boldsymbol{X}) = C - \log \|d\boldsymbol{X}\|$ where the $d\boldsymbol{X}$ is the required accuracy of the data. Therefore minimising the cost function C of ensemble learning is equivalent to minimising the coding lenght of the data.

3.2.4 Model Selection

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

$$p(\mathcal{H} \mid \boldsymbol{X}) = \frac{p(\boldsymbol{X} \mid \mathcal{H})p(\mathcal{H})}{p(\boldsymbol{X})}, \tag{3.13}$$

where $p(\mathcal{H})$ is the prior probability of a model and $p(\boldsymbol{X})$ is the probability of the data, which is constant. A lower bound on the evidence term $p(\boldsymbol{X} \mid \mathcal{H})$ is obtained from (3.8)

$$p(\mathbf{X} \mid \mathcal{H}) = \exp(C_{\mathrm{KL}} - C) \geq \exp(-C) \tag{3.14}$$

Multiple models can be used as a mixture of experts model [24]. The experts can be weighted with their probabilities given in equation (3.13). If the models have equal prior probabilities and the parameter approximations C_{KL} are equally good, the weights simplify to $\exp(-C)$. In practice, the costs tend to differ in the order of hundreds or thousands, which makes the model with the lowest cost C dominant. Therefore it is reasonable to concentrate on model selection rather than weighting.

3.3 Generalisation

A learning algorithm is said to *overlearn* the training data set, when its performance with test data starts to get worse during the learning with training data. The system starts to lose its ability to generalise. The same can happen when increasing the complexity of the model. The model is said to *overfit* to the data. When the model is too simple or the learning is stopped too early, the problem is called *underfitting* or *underlearning* accordingly. Balancing between over- and underfitting has perhaps been the main difficulty in model building.

There are ways to fight overfitting and overlearning [24, 5, 9]. Weight decay [41] for MLP networks corresponds to moving from ML to MAP solution in Figure 3.2. It punishes the model for using large values for weights and thus makes the mapping smoother. The same idea can be taken further - one can do model selection with MAP estimates by introducing a heuristical punishment term for model complexity. A popular and less heuristic method to select the best time to stop learning or the best complexity of a model is the cross-validation [24]. Part of the training data is left for validation and the models are compared based on their performance with the validation set.

Bayesian learning solves the tradeoff between under- and overfitting. If one has to select a single value for the parameter vector, it should represent the posterior probability mass well. The MAP solutions are attracted to high but sometimes narrow peaks. Figure 3.1 shows a situation, where search for the MAP solution first finds a good representative, but then moves to the highest peak which is on the border. This type of situation seems to be very common and the effect becomes stronger, when the dimensionality of the parameter space increases.

Because the KL divergence involves an expectation over a distribution, it is sensitive to the probability mass rather than to the probability density. Therefore ensemble learning is not attracted so much to narrow peaks and overfitting is largely avoided. Experiments have shown [57] that ensemble learning performs well in generalisation.

Two example cases are described in the following Sections. In the first simple

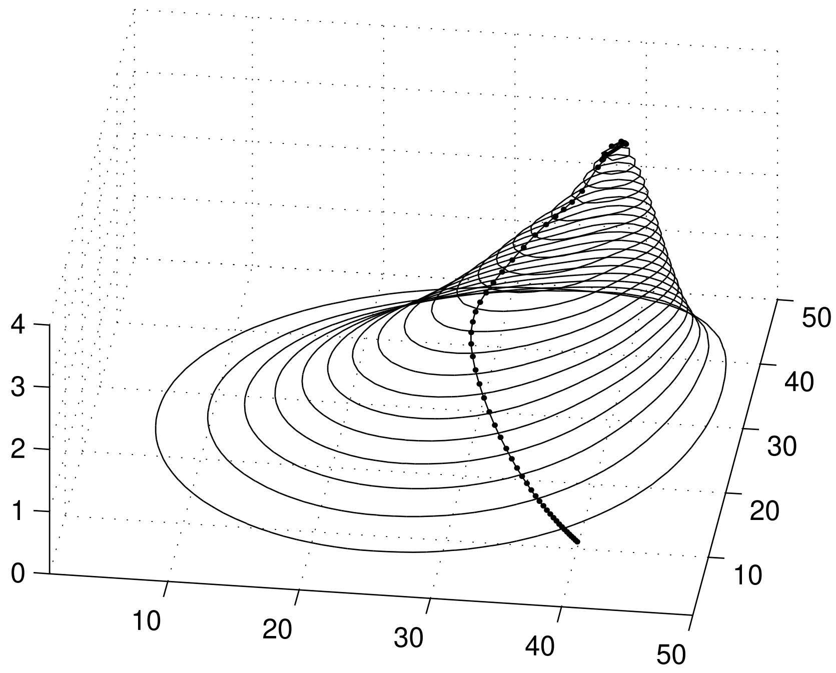


Figure 3.1: A hypothetical posterior pdf. A point estimate could first find a good representative of the probability mass, but then overfit to a narrow peak.

case, it is sufficient to use MAP estimate instead of ML to avoid overfitting. In the second example, however, both of the point estimates fail.

3.3.1 An Example on Polynomial Fitting

The data set $\boldsymbol{X}$ consists of 10 points on a plane. Model $\mathcal{H}$ states, that the points have been generated by a sixth order polynomial, whose weights are drawn from a Gaussian distribution with a zero mean and standard deviation (std) 2 and a Gaussian noise with std 0.1 is added. The problem is to find these weights.

Figure 3.2 shows the results. There are many different polynomials that fit quite well to the data. The ML solution does the fitting best, but the weights of the polynomial are large and the polynomial has a complicated form. The MAP solution takes the prior distributions into account, and the result is smoother. Bayesian learning takes into account all polynomials and weights them with their posterior probability. It solves the tradeoff between under-

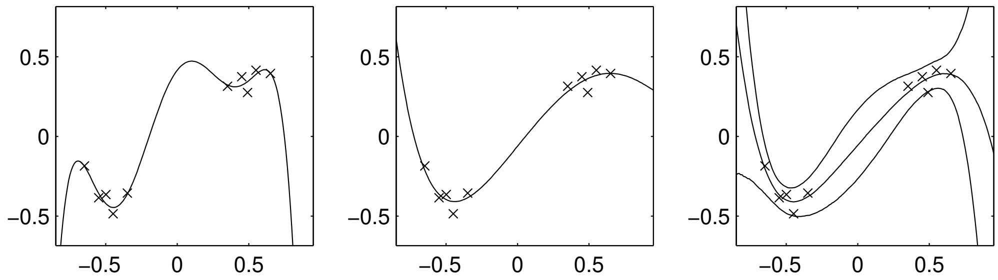


Figure 3.2: A sixth order polynomial is fitted to 10 data points. Left: Maximum likelihood solution. Middle: Maximum a posteriori solution. Right: Bayesian solution. The three curves present 5%, 50% and 95% fractiles.

and overfitting. Note that the error fractiles are closer in the parts of the polynomial that have data points.

3.3.2 Example Where Point Estimates Fail

The following example illustrates what can go wrong with point estimates. Three dimensional data vectors $\mathbf{x}(t)$ are modelled with linear factor analysis

$$\mathbf{x}(t) = \mathbf{A}s(t) + \mathbf{n}(t), \tag{3.15}$$

using a single source $s(t)$. The weight matrix $\mathbf{A}$ might get a value

$$\mathbf{A} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \tag{3.16}$$

while the source can just copy the values of the first dimension of $\mathbf{x}(t)$

$$s(t) = x_1(t). \tag{3.17}$$

The noise model is Gaussian with zero mean and parameterised variance

$$p(n_k) = \mathrm{N}\left(n_k; 0, \sigma_k^2\right). \tag{3.18}$$

When the reconstruction error or the noise term is evaluated

$$\mathbf{n}(t) = \mathbf{x}(t) - \mathbf{A}s(t) = \begin{bmatrix} 0 \\ x_2(t) \\ x_3(t) \end{bmatrix}, \tag{3.19}$$

one can see that problems will arise with the first variance parameter σ_1^2. The likelihood of the data

$$p(\mathbf{x}(t)) = p(\mathbf{n}(t)) = \prod_{k=1}^{3} \left(2\pi\sigma_k^2\right)^{-1/2} \exp\left(-\frac{x_k(t)^2}{2\sigma_k^2}\right) \tag{3.20}$$

$$= \left[\frac{1}{\sqrt{2\pi}} \prod_{k=2}^{3} \left(2\pi\sigma_k^2\right)^{-1/2} \exp\left(-\frac{x_k(t)^2}{2\sigma_k^2}\right)\right] \frac{1}{\sigma_1} \tag{3.21}$$

goes to infinity when the variance σ_1^2 goes to zero. Same applies to the posterior density, since it is basically just the likelihood multiplied by a finite factor.

The found model is completely useless and still, it is rated as infinitely good using point estimates. These problems are typical for models with estimates of the noise level or products. They can be sometimes avoided by fixing the noise level or using certain normalisations [2].

3.3.3 Role of Prior Information

Basically, all generalisation is based on the prior knowledge [47]. Training data provides information only at the data points and prior knowledge like piecewise smoothness of natural phenomena is needed for generalising to future data. This means that all learning has some priors that are either implicit or explicit as in Bayesian inference. In practice, the priors are often too strict.

Bayesian solutions are sensitive to the priors. Bad or wrong prior information can lead to too a complex model and in a sense overfitting [42]. The selection of appropriate priors requires lots of expert work and therefore so called *non-informative priors* [37] are searched for. They could be used in case of complete ignorance of the problem at hand. With hierarchical priors [18], some choices can be moved to higher levels which contain less information.

Chapter 4

Building Blocks for Hierarchical Nonlinear Factor Analysis

Hierarchical latent variable models can be constructed from simple building blocks. The basic idea is that adapting can be done locally. This provides a way to construct complicated models that can still be used with linear complexity. The formulas and implementation can be done for each block separately therefore reducing the possibility of errors and allowing more attention to other matters. It also increases the extensibility. In this chapter, the building blocks and equations for computation with them are introduced. The building blocks consist of variable nodes and computation nodes. The symbols for them are shown in Figure 4.1.

The network is described using terms of neural networks [24] and Bayesian networks [33] when applicable. The nodes are attached to each other using *signals*. Each node has input and output values or signals. For variable nodes, input is a value which is used for the prior distribution and output is the value of the variable. The variable nodes are continuous valued with a Gaussian prior. Each variable can be either observed or latent. Time dependent latent variables are called sources or neurons and time independent latent variables are called parameters or weights. For computation nodes, output is a fixed function of the inputs.

Since the variable nodes are probabilistic, the values propagated between the nodes have distributions. When ensemble learning together with a factorial posterior approximation is used, the cost function can be computed by propagating certain expected values instead of full distributions as can be seen in (4.1). Consequently the cost function can be minimised based on gradients

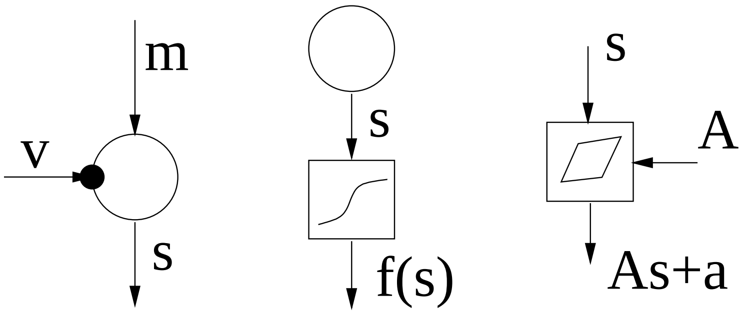


Figure 4.1: Left: A Gaussian latent variable s, marked with a circle, has a prior mean m and a prior variance $\exp(-v)$. Middle: A nonlinearity f is applied immediately after a Gaussian variable. Right: An affine transformation is made to the signal vector $\mathbf{s}$.

w.r.t. these expectations computed by back-propagation [24]. The gradients define the likelihood. Prior probabilities propagate forward, likelihoods propagate backward and they are combined to posterior probabilities.

The input for prior mean of a Gaussian node requires the mean $\langle\cdot\rangle$ and variance $\operatorname{Var}\{\cdot\}$. With a suitable parametrisation, mean $\langle\cdot\rangle$ and expected exponential $\langle\exp\cdot\rangle$ are required from the input for prior variance. The output of a Gaussian node can provide the mean $\langle\cdot\rangle$, variance $\operatorname{Var}\{\cdot\}$ and expected exponential $\langle\exp\cdot\rangle$ and can thus be used as an input to both the mean and variance of another Gaussian node. The expectations required by the inputs and provided by the outputs of different nodes are listed below:

	$\langle\cdot\rangle$	$\operatorname{Var}\{\cdot\}$	$\langle\exp\cdot\rangle$
Output provides:			
Gaussian	+	+	+
Gaussian with nonlinearity	+	+	
addition	+	+	+
multiplication	+	+	
Prior for variable nodes requires:			
mean of Gaussians	+	+	
variance of Gaussians	+		+

The variables can be gathered to vectors and matrices in a straigthforward manner. Other nodes that are compatible with the ones shown here can be found in [66].

4.1 Gaussian Variables

A Gaussian variable s has two inputs m and v and prior probability $p(s|m, v) = \mathrm{N}(s; m, \exp(-v))$. The variance is parameterised this way because then the mean and expected exponential of v suffice for computing the cost function. In Appendix A, it is shown that when s, m and v are mutually independent, i.e. $q(s, m, v) = q(s)q(m)q(v)$, $C_{s,p} = -\langle \ln p(s|m, v)\rangle$ yields

$$C_{s,p} = \frac{1}{2}\left\{\langle \exp v\rangle \left[(\overline{s} - \langle m\rangle)^2 + \mathrm{Var}\{m\} + \widetilde{s}\right] - \langle v\rangle + \ln 2\pi\right\}. \tag{4.1}$$

For observed variables this is the only term in the cost function but for latent variables there is also $C_{s,q}$: the part resulting from $\langle \ln q(s)\rangle$. The posterior approximation $q(s)$ is defined to be Gaussian with mean $\overline{s}$ and variance $\widetilde{s}$: $q(s) = \mathrm{N}(s; \overline{s}, \widetilde{s})$. This yields

$$C_{s,q} = -\frac{1}{2}\ln 2\pi e\widetilde{s} \tag{4.2}$$

which is the negative entropy of Gaussian variable with variance $\widetilde{s}$. The parameters $\overline{s}$ and $\widetilde{s}$ are to be optimised during learning.

The output of a latent Gaussian node trivially provides expectation and variance: $\langle s\rangle = \overline{s}$ and $\mathrm{Var}\{s\} = \widetilde{s}$. The expected exponential is

$$\begin{aligned}
\langle \exp s\rangle &= \int q(s)e^s ds & (4.3)\\
&= \int (2\pi\widetilde{s})^{-1/2}\exp\left[\frac{-(s-\overline{s})^2}{2\widetilde{s}} + s\right] ds & (4.4)\\
&= \int (2\pi\widetilde{s})^{-1/2}\exp\left[\frac{-(s-\overline{s}-\widetilde{s})^2}{2\widetilde{s}} + \overline{s} + \frac{\widetilde{s}}{2}\right] ds & (4.5)\\
&= \exp(\overline{s} + \widetilde{s}/2). & (4.6)
\end{aligned}$$

The outputs of observed nodes are scalar values instead of distributions and thus $\langle s\rangle = s$, $\mathrm{Var}\{s\} = 0$ and $\langle \exp s\rangle = \exp s$.

4.1.1 Update Rule

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

1. First, the gradients of C_p w.r.t. $\langle s\rangle$, $\mathrm{Var}\{s\}$ and $\langle \exp s\rangle$ are computed.

2. Second, the terms in C_p which depend on $\overline{s}$ and $\widetilde{s}$ are assumed to be $a\overline{s} + b[(\overline{s} - \overline{s}_{\text{current}})^2 + \widetilde{s}] + c\langle \exp s \rangle + d$, where $a = \partial C_p / \partial \overline{s}$, $b = \partial C_p / \partial \widetilde{s}$ and $c = \partial C / \partial \langle \exp s \rangle$. This is shown to be true in Section 4.5.

3. Third, the minimum of $C_s = C_{s,p} + C_{s,q}$ is solved.

The cost function is written explicitely as the function of $\overline{s}$ and $\widetilde{s}$:

$$C_s(\overline{s}, \widetilde{s}) = a_1 \overline{s}^2 + a_2 \overline{s} + a_3 \widetilde{s} + a_4 \ln \widetilde{s} + c \exp(\overline{s} + \widetilde{s}/2) + a_5, \tag{4.7}$$

where $a_1 > 0$ and $a_4 < 0$. First, $\widetilde{s}$ is kept constant and the optimal $\overline{s}$ is solved using Newton's iteration. Then the optimal $\widetilde{s}$ is solved using a stabilised fixed-point iteration and keeping $\overline{s}$ constant.

In the special case $c = 0$, the minimum of $C_s(\overline{s}, \widetilde{s})$

$$\overline{s}_{\text{opt}} = -\frac{a_2}{2a_1} \tag{4.8}$$

$$\widetilde{s}_{\text{opt}} = \frac{a_4}{a_3} \tag{4.9}$$

can be found analytically. In this case, $q(s)$ is optimal among all functions i.e. the free-form approximation.

4.1.2 First Example

The purpose of this first example is to illustrate what is meant with the propagation of expected values and derivatives. The model structure is shown in Figure 4.2. The scalar observed data $x(t)$ is a Gaussian variable with prior mean m and prior variance $\exp(-v)$:

$$p(x(t) \mid m) = \mathrm{N}(x(t); m, \exp(-v)). \tag{4.10}$$

The only two parameters in the model are latent Gaussian variables m and v:

$$p(m) = \mathrm{N}(m; 0, \exp(-(-5))) \tag{4.11}$$

$$p(v) = \mathrm{N}(v; 0, \exp(-(-5))) \tag{4.12}$$

with fixed prior means and variances.

The model basically states that the data points are scattered with an unknown mean and variance. They can be estimated from the data. An estimate of the posterior distributions $p(m \mid \boldsymbol{X}, \mathcal{H})$ and $p(v \mid \boldsymbol{X}, \mathcal{H})$ are

$$q(m) = \mathrm{N}(m; \overline{m}, \widetilde{m}) \tag{4.13}$$

$$q(v) = \mathrm{N}(v; \overline{v}, \widetilde{v}), \tag{4.14}$$

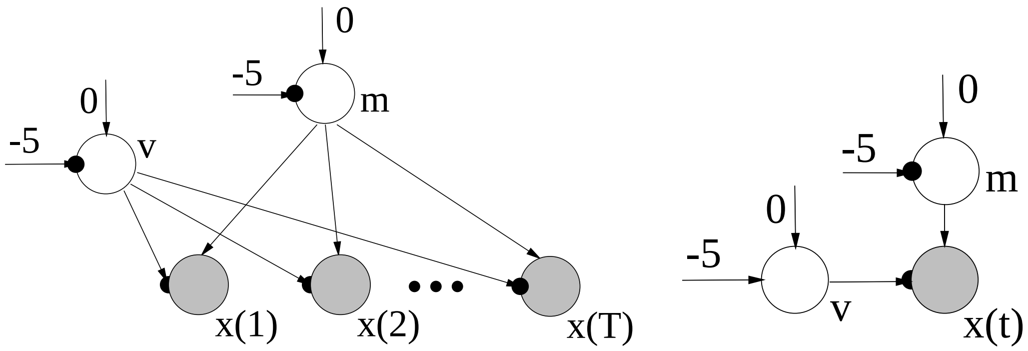


Figure 4.2: Left: A very simple example structure: There are two Gaussian variables m and v that generate the distribution of the data $x(t)$. Right: The same structure is visualised using $x(t)$ to represent all observations $t = 1, 2, \ldots, T$.

where $\overline{m}$, $\widetilde{m}$, $\overline{v}$ and $\widetilde{v}$ are the parameters controlling the posterior approximation.

The expected value and the variance are required from the prior mean m of $x(t)$. The expected value and the expected exponential are required from the variable v that determines the prior variance of $x(t)$. The outputs of Gaussian variables can provide all of these expected values.

The learning would start with some initial values, say $\overline{m} = 0$, $\widetilde{m} = 1$, $\overline{v} = 0$ and $\widetilde{v} = 1$. The cost concerning the observations $x(t)$ from (4.1) is

$$C_x = \sum_{t=1}^{T} \frac{1}{2} \left\{ \langle \exp v \rangle \left[(x(t) - \overline{m})^2 + \widetilde{m} \right] - \overline{v} + \ln 2\pi \right\} \tag{4.15}$$

and its partial derivatives with respect to m

$$\frac{\partial C_x}{\partial \langle m \rangle} = \langle \exp v \rangle \sum_{t=1}^{T} [\overline{m} - x(t)] \tag{4.16}$$

$$\frac{\partial C_x}{\partial \mathrm{Var}\{m\}} = \frac{T \langle \exp v \rangle}{2} \tag{4.17}$$

are propagated upwards to the variable node m. There the C_x is assumed to be of the form $C_x = a \langle m \rangle + b[(\langle m \rangle - \langle m \rangle_{\text{current}})^2 + \mathrm{Var}\{m\}] + c \langle \exp m \rangle + d$, which indeed is true.

The part of the cost function that concerns the variable m directly has two parts simplified from Equations (4.1) and (4.2)

$$C_{m,p} = \frac{1}{2}\left[\langle \exp(-5)\rangle\left(\overline{m}^2 + \widetilde{m}\right) - \langle -5\rangle + \ln 2\pi\right] \tag{4.18}$$

$$C_{m,q} = -\frac{1}{2}\ln 2\pi e\widetilde{m}. \tag{4.19}$$

The parameters $\overline{m}$ and $\widetilde{m}$ can now be updated so that the total affected cost $C_x + C_{m,p} + C_{m,q}$ is minimised. The minimum can be obtained analytically in this case and it is

$$\overline{m} = \frac{\sum_{t=1}^{T} x(t)}{T + \exp(-5)/\langle \exp v\rangle} \tag{4.20}$$

$$\widetilde{m} = \frac{1}{T\langle \exp v\rangle + \exp(-5)}. \tag{4.21}$$

From the result we can see that the mean of m is close to the mean of the observations. The prior in (4.11) pulls it slightly towards zero. The uncertainty of the prior mean $\widetilde{m}$ does not directly depend on the actual data. When the number of observations T increases, the uncertainty i.e. the posterior variance $\widetilde{m}$ decreases towards zero.

The partial derivatives of C_x defined in 4.15 with respect to v are

$$\frac{\partial C_x}{\partial \langle v\rangle} = \sum_{t=1}^{T}\frac{1}{2}\left[(x(t) - \overline{m})^2 + \widetilde{m}\right] \tag{4.22}$$

$$\frac{\partial C_x}{\partial \langle \exp v\rangle} = -\frac{T}{2}. \tag{4.23}$$

There are also the terms similar to (4.18) and (4.19). Therefore, when optimising $\overline{v}$ and $\widetilde{v}$, one has to minimise a more complicated function of $\overline{v}$ and $\widetilde{v}$. It must be done iteratively.

The optimal solution for $q(m)$ depends on $q(v)$ and vice versa. This means that to fully solve the problem, one could update $q(m)$ and $q(v)$ alternately.

4.2 Addition

Addition and multiplication nodes can be used e.g. for constructing affine transformations between the variables. Denoting the inputs by s_k, the output is $\sum_k s_k$ for an addition node. The mean, variance and expected exponential of the addition node are

$$
\begin{aligned}
\langle s_1 + s_2 \rangle &= \langle s_1 \rangle + \langle s_2 \rangle && (4.24)\\
\mathrm{Var}\{s_1 + s_2\} &= \mathrm{Var}\{s_1\} + \mathrm{Var}\{s_2\} && (4.25)\\
\langle \exp(s_1 + s_2) \rangle &= \langle \exp s_1 \rangle \langle \exp s_2 \rangle && (4.26)
\end{aligned}
$$

assuming that the variables s_k are independent. The derivatives of the cost function propagate to the inputs using the chain rule applied to each of the Equations (4.24), (4.25) and (4.26). They are

$$
\begin{aligned}
\frac{\partial C}{\partial \langle s_1 \rangle} &= \frac{\partial C}{\partial \langle s_1 + s_2 \rangle} \frac{\partial \langle s_1 + s_2 \rangle}{\partial \langle s_1 \rangle} = \frac{\partial C}{\partial \langle s_1 + s_2 \rangle} && (4.27)\\
\frac{\partial C}{\partial \mathrm{Var}\{s_1\}} &= \frac{\partial C}{\partial \mathrm{Var}\{s_1 + s_2\}} \frac{\partial \mathrm{Var}\{s_1 + s_2\}}{\partial \mathrm{Var}\{s_1\}} = \frac{\partial C}{\partial \mathrm{Var}\{s_1 + s_2\}} && (4.28)\\
\frac{\partial C}{\partial \langle \exp s_1 \rangle} &= \frac{\partial C}{\partial \langle \exp(s_1 + s_2) \rangle} \frac{\partial \langle \exp(s_1 + s_2) \rangle}{\partial \langle \exp(s_1) \rangle} && (4.29)\\
&= \langle \exp s_2 \rangle \frac{\partial C}{\partial \langle \exp(s_1 + s_2) \rangle}.
\end{aligned}
$$

The equations for larger sums are obtained by induction, e.g. $s_1 + s_2 + s_3 = (s_1 + s_2) + s_3$.

4.3 Multiplication

Multiplication node has inputs s_k and the output is their product $\prod_k s_k$. The excepted exponential of the output cannot be evaluated without knowing the exact distribution of the inputs. Assuming independence between s_k, the mean and the variance of the output are

$$
\begin{aligned}
\langle s_1 s_2 \rangle &= \langle s_1 \rangle \langle s_2 \rangle && (4.30)\\
\mathrm{Var}\{s_1 s_2\} &= \left\langle s_1^2 s_2^2 \right\rangle - \langle s_1 s_2 \rangle^2 \\
&= \left\langle s_1^2 \right\rangle \left\langle s_2^2 \right\rangle - \langle s_1 \rangle^2 \langle s_2 \rangle^2 && (4.31)\\
&= (\langle s_1 \rangle^2 + \mathrm{Var}\{s_1\})(\langle s_2 \rangle^2 + \mathrm{Var}\{s_2\}) - \langle s_1 \rangle^2 \langle s_2 \rangle^2 \\
&= \langle s_1 \rangle^2 \mathrm{Var}\{s_2\} + \mathrm{Var}\{s_1\} \left(\langle s_2 \rangle^2 + \mathrm{Var}\{s_2\}\right). && (4.32)
\end{aligned}
$$

The propagated derivatives are obtained like with addition and they are

$$\begin{aligned}
\frac{\partial C}{\partial \langle s_1 \rangle} &= \frac{\partial C}{\partial \langle s_1 s_2 \rangle} \frac{\partial \langle s_1 s_2 \rangle}{\partial \langle s_1 \rangle} + \frac{\partial C}{\partial \mathrm{Var}\{s_1 s_2\}} \frac{\partial \mathrm{Var}\{s_1 s_2\}}{\partial \langle s_1 \rangle} && (4.33)\\
&= \langle s_2 \rangle \frac{\partial C}{\partial \langle s_1 s_2 \rangle} + 2\mathrm{Var}\{s_2\} \frac{\partial C}{\partial \mathrm{Var}\{s_1 s_2\}} \langle s_1 \rangle \\
\frac{\partial C}{\partial \mathrm{Var}\{s_1\}} &= \frac{\partial C}{\partial \langle s_1 s_2 \rangle} \frac{\partial \langle s_1 s_2 \rangle}{\partial \mathrm{Var}\{s_1\}} + \frac{\partial C}{\partial \mathrm{Var}\{s_1 s_2\}} \frac{\partial \mathrm{Var}\{s_1 s_2\}}{\partial \mathrm{Var}\{s_1\}} && (4.34)\\
&= \left(\langle s_2 \rangle^2 + \mathrm{Var}\{s_2\}\right) \frac{\partial C}{\partial \mathrm{Var}\{s_1 s_2\}}.
\end{aligned}$$

The equations for larger products are again obtained by induction, e.g. $s_1 s_2 s_3 = (s_1 s_2) s_3$.

4.4 Gaussian Variable with Nonlinearity

A nonlinear computation node can be used for constructing nonlinear mappings between the variable nodes. The output of the nonlinearity is a fixed nonlinear function of the input. The nonlinearity is always used immediately after a Gaussian variable. This modifies the way the variable is handled.

4.4.1 Nonlinearities

Figure 4.3 shows some possible transfer functions. A good nonlinearity should have an area that is close to linear and another that is saturated. Linear part guarantees the possibility to use the unit as a linear one. Saturated area makes sparse representation possible as was seen in Section 2.3. When the input fluctuates inside the the saturated area, the output does not effectively vary. Thus, the input does not always need to be precisely determined for the output to be precise. A nonlinearity with two flat parts can be used as a binary unit that has two typical output values and almost nothing in between.

Some expected values after the nonlinearity can be evaluated for Gaussian input [20]. Therefore we restrict the nonlinearity to follow immediately after a Gaussian variable node. Now the mean, variance and the expected exponential of the output of a nonlinearity have integral expressions. For most nonlinear functions it is impossible to compute them analytically, but for the function $f(s) = \exp(-s^2)$ the mean and variance do have analytical expressions. Therefore it is used in this work. The required expectations of the

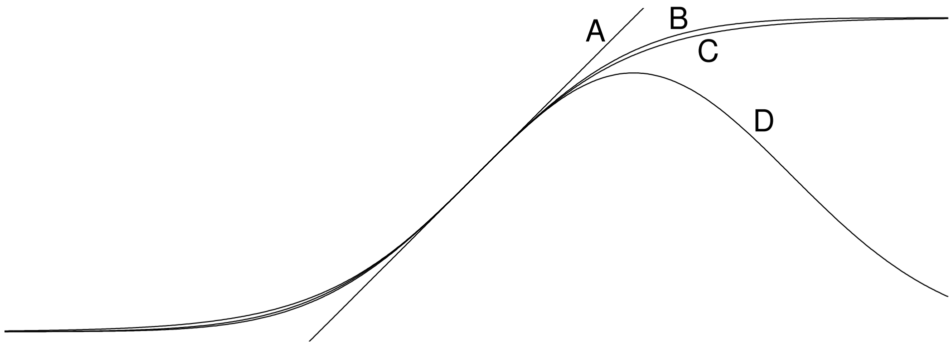


Figure 4.3: Transfer functions are scaled such that zeros of the second derivatives coincide. A) linear $f(s) = s$, B) error function $f(s) = \int_{-\infty}^{s} \exp(-r^2)dr$, C) sigmoid or the logistic function $f(s) = (1 + e^{-s})^{-1}$, D) Gaussian $f(s) = \exp(-s^2)$. The Gaussian lies between the other two nonlinear functions in the left part.

outputs are

$$\langle f(s) \rangle = \exp\left(-\frac{\overline{s}^2}{2\widetilde{s}+1}\right)(2\widetilde{s}+1)^{-\frac{1}{2}} \tag{4.35}$$

$$\left\langle f(s)^2 \right\rangle = \exp\left(-\frac{2\overline{s}^2}{4\widetilde{s}+1}\right)(4\widetilde{s}+1)^{-\frac{1}{2}}. \tag{4.36}$$

The variance is obtained by $\mathrm{Var}\left\{f(s)\right\} = \langle f^2(s) \rangle - \langle f(s) \rangle^2$.

Gaussian radial basis functions (RBF) [24] use the same nonlinearity but the input is the distance from a certain point in the source space rather than one of the sources directly. Ghahramani and Roweis [20] used Gaussian RBF approximators with EM algorithm to model nonlinear dynamical systems. They found that using the Gaussian nonlinearity the integrals become tractable. Another potential possibility would be to use the error function $f(s) = \int_{-\infty}^{s} \exp(-r^2)dr$, since the mean can be evaluated analytically and the variance can be approximated from above [15]. This is useful, since increasing the variance increases also the cost function and minimising an upper bound for the cost guarantees it to be low. Murphy [50] used the logistic function approximated iteratively with a Gaussian. Valpola [62] approximated the same function with a truncated Taylor series.

Hornik [26] and Funahashi [17] have independently shown that MLP networks are universal approximators, that is, given enough hidden units the mapping from inputs to outputs can approximate any measurable function to any desired degree of accuracy. This result was proven for any non-decreasing nonlinearity $f(s)$ that has the limits $\lim_{s\to-\infty} f(s) = 0$ and $\lim_{s\to\infty} f(s) = 1$. Unfortunately, that is not true for the function $f(s) = \exp(-s^2)$. Future work might include a comparison with other nonlinearities and perhaps the property of universal approximation could be proven at least for a finite interval.

4.4.2 Update Rule

The update of a Gaussian node followed by the nonlinearity is similar to the plain Gaussian node. The source is updated to minimise the terms of the cost function defined in (3.8), that are affected. Other parts of the network are considered constant during the update.

In addition to the terms arising from the variable itself defined in (4.1) and (4.2), the terms corresponding to the variables that the output is propagated to are affected. The gradients of C_p w.r.t. $\langle f(s)\rangle$ and $\mathrm{Var}\{f(s)\}$ are assumed to arise from a quadratic term $a\langle f(s)\rangle + b[(\langle f(s)\rangle - \langle f(s)\rangle_{\text{current}})^2 + \mathrm{Var}\{f(s)\}] + d$. This assumption is shown to be true in Section 4.5.

The update is done by repeating the following steps until they are shorter than some very small constant value.

1. First, the cost function and the gradients of it w.r.t. $\overline{s}$ and $\widetilde{s}$ are computed.
2. Second, update candidates for $\widetilde{s}$ and $\overline{s}$ are found using is a fixed point iteration and an approximate Newton's method accordingly.
3. Third, the candidates are tested by computing the cost function again and the step size is halved as long as the cost function is about to increase.

The formulas and the proof that the cost decreases or it has converged, can be found in Appendix B.

4.5 Form of the Cost Function

The form of the part of the cost function that an output of a neuron affects is shown to be of the form $a\langle\cdot\rangle + b[(\langle\cdot\rangle - \langle\cdot\rangle_{\text{current}})^2 + \mathrm{Var}\{\cdot\}] + c\langle\exp\cdot\rangle + d$. If the output is connected directly to another variable, this can be seen directly from

Equation (4.1). When the output is connected to multiple variables, the sum of the affected costs is of the same form. Now one has to prove that the form stays the same when the signals are fed through addition and multiplication nodes.

If the cost function has the predefined form for $s_1 + s_2$, it has the same form for s_1, when regarding s_2 constant. This can be shown using (4.24) and (4.25):

$$\begin{aligned} C &= a\langle s_1+s_2\rangle + b\left[(\langle s_1+s_2\rangle - \langle s_1+s_2\rangle_{\text{current}})^2 + \mathrm{Var}\{s_1+s_2\}\right] \\ &\quad + c\langle \exp(s_1+s_2)\rangle + d \qquad (4.37) \\ &= a\langle s_1\rangle + b\left[(\langle s_1\rangle - \langle s_1\rangle_{\text{current}})^2 + \mathrm{Var}\{s_1\}\right] \\ &\quad + (c\langle \exp s_2\rangle)\langle \exp s_1\rangle + (d + a\langle s_2\rangle + b\mathrm{Var}\{s_2\})\,. \end{aligned}$$

It can also be seen from (4.37) that when $c = 0$ for the sum s_1+s_2, it is zero for the addend s_1. This means that the outputs of product and nonlinear nodes can be fed through addition nodes.

If the cost function is of the predefined form with $c = 0$ for the product $s_1 s_2$, it is similar for s_1, when regarding s_2 constant. This can be shown using (4.30, 4.32)

$$\begin{aligned} C &= a\langle s_1 s_2\rangle + b\left[(\langle s_1 s_2\rangle - \langle s_1 s_2\rangle_{\text{current}})^2 + \mathrm{Var}\{s_1 s_2\}\right] + d \qquad (4.38) \\ &= (a\langle s_2\rangle + 2b\mathrm{Var}\{s_2\}\langle s_1\rangle_{\text{current}})\langle s_1\rangle \\ &\quad + \left[b\left(\langle s_2\rangle^2 + \mathrm{Var}\{s_2\}\right)\right]\left[(\langle s_1\rangle - \langle s_1\rangle_{\text{current}})^2 + \mathrm{Var}\{s_1\}\right] \\ &\quad + \left(d - b\mathrm{Var}\{s_2\}\langle s_1\rangle^2_{\text{current}}\right)\,. \end{aligned}$$

When one calculates the partial derivatives of a cost of this form

$$C_p = a\overline{s} + b[(\overline{s} - \overline{s}_{\text{current}})^2 + \widetilde{s}] + c\langle \exp s\rangle + d, \qquad (4.39)$$

one finds out that they are simply

$$\frac{\partial C_p}{\partial \overline{s}} = a \qquad (4.40)$$

$$\frac{\partial C_p}{\partial \widetilde{s}} = b \qquad (4.41)$$

$$\frac{\partial C}{\partial \langle \exp s\rangle} = c. \qquad (4.42)$$

Chapter 5

Hierarchical Nonlinear Factor Analysis with Variance Modelling

This chapter describes a model called hierarchical nonlinear factor analysis with variance modelling (HNFA+VM). It is built hierarchically from the building blocks described in Chapter 4. In addition to analysing nonlinear factors it can model variance dependencies of them. HNFA+VM is thus related to e.g. topographical ICA [30] that has been applied to analysing natural images with success.

The building blocks can be connected together rather freely but there are the following restrictions:

1. The resulting network has to be a directed acyclic graph so that the probability distributions would be normalisable [51].
2. Nonlinearity is always immediately after a Gaussian latent variable since the expectations are solved analytically only with a Gaussian input.
3. Outputs of multiplication or nonlinearity cannot propagate to a variance prior because the expected exponential cannot be evaluated.
4. There should be only one computational path from a latent variable to a variable. This assumption is used e.g. in (4.1), (4.25), (4.26), (4.30) and (4.31). If there are multiple paths, ensemble learning becomes more complicated [43] and the situation is out of the scope of this thesis.

The model introduced in this chapter has no model for dynamics. The time dependent variables are connected only to those time dependent variables that have the same time index. This means that dependencies in time are ignored and nothing would change, if the time indices were permutated. Time dependencies are an important direction for future work as will be discussed in Chapter 9.

Section 2.2.5 described nonlinear factor analysis (NFA) [43, 64]. The basic idea behind HNFA is to replace the deterministic hidden units or computational units of an MLP-like network in NFA by stochastic latent variables. The computational complexity of NFA is quadratic w.r.t. the number of nodes due to the different paths between sources and observations. In case of HNFA, the dependencies are broken off at the latent variables of the middle layer. This results in a linear computational complexity which is important for scalability. Different paths are now considered independent which means that the posterior approximation is less accurate. Not taking the dependencies into account increases the cost. The noise model in the hidden nodes in HNFA means that the reconstructions of the data do not have to be decided precisely at the uppermost layer in HNFA. Instead, the uppermost layers can just guide the lower ones which decide the actual reconstructions.

5.1 Variance Neurons

In most currently used models, the means of Gaussian nodes have hierarchical or dynamical models. In many real cases the variance is not constant either but it is more difficult to model it (see Section 3.3.2). A variance neuron shown in Figure 5.1 is designed for that. It can convert a prediction of mean into a prediction of variance and thus allows to build hierarchical or dynamical models for the variance. In general, the variance neuron results in a heavy-tailed super-Gaussian model for the Gaussian node it is attached to as described in Section 2.3. This can be useful for instance in modelling outliers in the observations.

In many cases, the amount of noise depends on the amplitude of a signal. To model that, one would like to connect the same variable to both the mean and variance priors of a neuron. Unfortunately, this is not allowed because of the fourth restriction concerning the independence assumptions. Instead, one can attach the variable to the mean inputs of a neuron $s(t)$ and its variance neuron $u(t)$. This effectively results in a correlation between the mean and the variance of $s(t)$. The addition of the variance neuron in between restores the

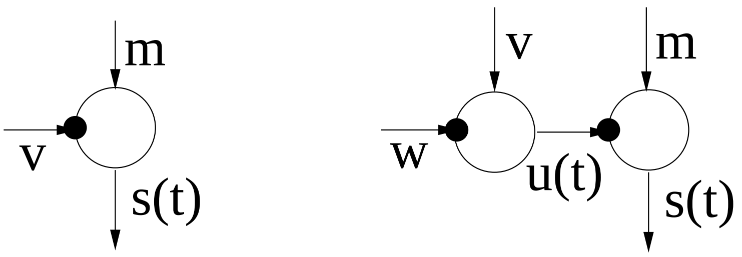


Figure 5.1: Left: Source $s(t)$ has a time independent prior variance v. Right: A variance neuron is included to give a time dependent prior variance $u(t)$ for the source $s(t)$.

formal independency at the expense of increased cost.

5.2 Formulation of the Model Structure

The nonlinear factor analysis described in Section 2.2.5 can in theory model any kind of probability density in the data space. If the hidden units or computational nodes of this MLP-like network are replaced by latent variables, one gets a hierarchical presentation, where the number of layers is not restricted. The learning or the adjustments can be done layer by layer in linear computational complexity. Between two adjacent layers the mapping can be quite simple and therefore easy to learn. Still, the total mapping through all the layers can be strongly nonlinear.

The visual cortex of mammals seems to have a hierarchical structure [27], which can perhaps be modelled with a similar hierarchical structure. Simple cells in the primary visual cortex (V1) show rectangular antagonistic on/off zones responding to bars of a particular orientation. Complex cells respond either to an edge, a bar or a slit stimulus of a particular orientation falling anywhere in its receptive field. The exact location of the stimulus within the receptive field is not as critical. The cells in the third category are called hypercomplex.

Dependencies between certain variances have been found [67] from image data. This could be taken into account in the model using variance neurons. Several higher-order statistical properties of natural images and signals can be explained by a stochastic model which simply varies scale of an otherwise stationary Gaussian process [55]. The independent components of images resemble features that simple cells respond to. In the independent subspace analysis, phase shift invariant features emerged [29], which corresponds to behaviour of complex cells. The difference to ICA is the correlation of variances of the features. Also there have been good results in using topographical ICA [30] on image data. It is also based on modelling correlations of variances of 'independent' components.

5.2.1 Main Structure

Figure 5.2 shows the structure for hierarchical nonlinear factor analysis with variance modelling (HNFA+VM). It utilises variance neurons and nonlinearities in building a hierarchical model for both the means and variances. Without the variance neurons the model would correspond to a multi-layer perceptron with latent variables as hidden neurons. Note that computational nodes as hidden neurons would result in multiple paths from upper layer latent variables to the observations. This type of structure was used in [43] and it has a quadratic as opposed to linear computational complexity.

The exact formulation of HNFA+VM is as follows. The observed data matrix $\boldsymbol{X}$ has T observations of n_1 dimensions. $\boldsymbol{X}$ is named $\mathbf{s}_1(t)$ for notational simplicity

$$\boldsymbol{X} = [\mathbf{s}_1(1), \mathbf{s}_1(2), \ldots, \mathbf{s}_1(T)] \,, \tag{5.1}$$

where $t \in \{1, 2, \ldots, T\}$ corresponds to different observations and the subscript 1 corresponds to the first layer.

On each layer i, there are n_i sources assembled to a vector $\mathbf{s}_i$. The dimensions of the vectors are marked with $s_{i,k}$, $k \in \{1, 2, \ldots, n_i\}$. The sources on upper layers $i > 1$ are mapped through a Gaussian nonlinearity

$$\mathbf{f}(\mathbf{s}_i(t)) = \begin{bmatrix} \exp(-s_{i,1}(t)^2) \\ \exp(-s_{i,2}(t)^2) \\ \vdots \\ \exp(-s_{i,n_i}(t)^2) \end{bmatrix} . \tag{5.2}$$

The connection downwards after the nonlinearity is done using the affine mappings

$$\mathbf{m}_i^s(t) = \begin{cases} \mathbf{A}_i \mathbf{f}(\mathbf{s}_{i+1}(t)) + \boldsymbol{\mu}_{s,i} & \text{if } i < n \\ \boldsymbol{\mu}_{s,i} & \text{if } i = n \end{cases} \tag{5.3}$$

$$\mathbf{m}_i^u(t) = \begin{cases} \mathbf{B}_i \mathbf{f}(\mathbf{s}_{i+1}(t)) + \boldsymbol{\mu}_{u,i} & \text{if } i < n \\ \boldsymbol{\mu}_{u,i} & \text{if } i = n \end{cases} , \tag{5.4}$$

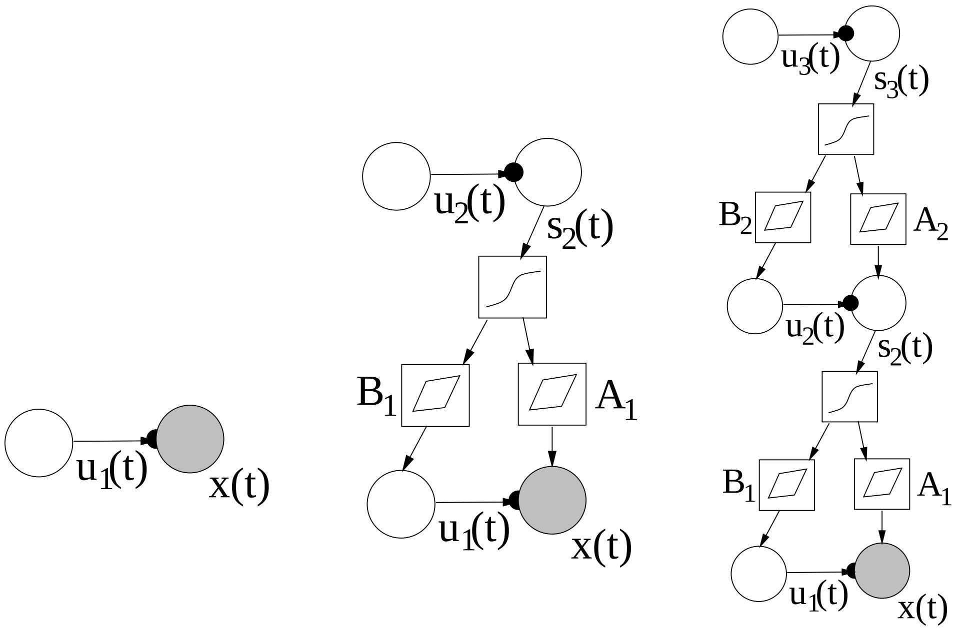


Figure 5.2: HNFA+VM model can be built up in stages. Left: A variance neuron is attached to each Gaussian observation node. The nodes represent vectors. Middle: A layer of sources with variance neurons attached to them is added. The nodes next to the weight matrices $\mathbf{A}_1$ and $\mathbf{B}_1$ represent affine transformations including a bias term. Right: Another layer is added. The size of the layers may vary. More layers can be added in the same manner. Note that some parameters are left out of the picture for clarity.

where n is the number of layers.

Each source $s_{i,k}$ has a corresponding variance neurons $u_{i,k}$. The signals $\mathbf{m}_i^s(t)$ and $\mathbf{m}_i^u(t)$ are used as prior means for them

$$
\begin{aligned}
p(s_{i,k}(t) \mid \mathbf{s}_{i+1}(t), u_{i,k}(t), \dots) &= \mathrm{N}\left(s_{i,k}(t); m_{i,k}^s(t), \exp(-u_{i,k}(t))\right) && (5.5)\\
p(u_{i,k}(t) \mid \mathbf{s}_{i+1}(t), \dots) &= \mathrm{N}\left(u_{i,k}(t); m_{i,k}^u(t), \exp(-\sigma_{u,i,k})\right). && (5.6)
\end{aligned}
$$

The prior variance of source vector $\mathbf{s}_i(t)$ is the corresponding variance source vector $\mathbf{u}_{i,k}(t)$ and the prior variance of variance sources is a parameter vector $\boldsymbol{\sigma}_{u,i}$.

5.2.2 Hierarchical Prior for the Weights

The priors for weights $\mathbf{A}$ and $\mathbf{B}$ have zero mean and common variance to each dimension k, which corresponds to incoming weights of a source $s_{i,k}$:

$$p(A_{i,k,j} \mid v^A_{i,j}) = \mathrm{N}\left(A_{i,k,j}; 0, \exp(-v^A_{i,j})\right), \tag{5.7}$$

where $A_{i,k,j}$ is the element of $\mathbf{A}_i$ that connects $s_{i+1,j}$ to $s_{i,k}$. Parameters for $\mathbf{B}$ are similar and are not shown here. The variance parameter has the prior parameters that are common to dimensions j

$$p(v^A_{i,j} \mid m^{vA}_i, v^{vA}_i) = \mathrm{N}\left(v^A_{i,j}; m^{vA}_i, \exp(-v^{vA}_i)\right). \tag{5.8}$$

The hyperparameters m^{vA}_i and v^{vA}_i have priors of the form

$$p(m^{vA}_i \mid m^{mvA}_i, v^{mvA}_i) = \mathrm{N}\left(m^{vA}_i; m^{mvA}_i, \exp(-v^{mvA}_i)\right). \tag{5.9}$$

The priors for these priors m^{mvA}_i, m^{vvA}_i, v^{mvA}_i and v^{vvA}_i of the hyperparameters are very flat and defined as constants in the model structure. They are of the form

$$p(m^{mvA}_i) = \mathrm{N}\left(m^{mvA}_i; 0, 100^2\right). \tag{5.10}$$

5.2.3 Hierarchical Prior for the Sources

The hierarchical structure of the parameters concerning the sources $\mathbf{s}$ and $\mathbf{u}$ is somewhat similar to that of the weights. The parameters are

$$\begin{aligned}
p(\mu_{s,i,k} \mid m^{ms}_{i,k}, v^{ms}_{i,k}) &= \mathrm{N}\left(\mu_{s,i,k}; m^{ms}_{i,k}, \exp(-v^{ms}_{i,k})\right) && (5.11)\\
p(\mu_{u,i,k} \mid m^{mu}_{i,k}, v^{mu}_{i,k}) &= \mathrm{N}\left(\mu_{u,i,k}; m^{mu}_{i,k}, \exp(-v^{mu}_{i,k})\right) && (5.12)\\
p(\sigma_{u,i,k} \mid m^{vu}_{i,k}, v^{vu}_{i,k}) &= \mathrm{N}\left(\sigma_{u,i,k}; m^{vu}_{i,k}, \exp(-v^{vu}_{i,k})\right). && (5.13)
\end{aligned}$$

The hyperparameters have priors that are common to different components k of a parameter vector

$$p(m^{ms}_{i,k} \mid m^{mms}_i, v^{mms}_i) = \mathrm{N}\left(m^{ms}_{i,k}; m^{mms}_i, \exp(-v^{mms}_i)\right) \tag{5.14}$$

and similarly for $v^{ms}_{i,k}$, $m^{mu}_{i,k}$, $v^{mu}_{i,k}$, $m^{vu}_{i,k}$ and $v^{vu}_{i,k}$. The priors for the hyperparameters are again flat and fixed

$$\begin{aligned}
p(m^{mms}_i) &= \mathrm{N}\left(m^{mms}_i; 0, 100^2\right) && (5.15)\\
p(v^{mms}_i) &= \mathrm{N}\left(v^{mms}_i; 0, 100^2\right) && (5.16)
\end{aligned}$$

and so forth. The scale of the data should be made small enough as a preprocessing step.

5.2.4 Comparison of the Notation

The more theoretical notation of Chapter 3 can be fit to the notation in this chapter. The model structure $\mathcal{H}$ corresponds to the definitions, the number of layers, sizes of the layers and the uppermost fixed variances 100^2. The data $\boldsymbol{X}$ does not differ since it can be found in equation (5.1). The parameters $\boldsymbol{\theta}$ correspond to the sources and the parameters marked with $\mathbf{s}, \mathbf{u}, \mathbf{A}, \mathbf{B}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \mathbf{m}$ and $\mathbf{v}$.

5.3 Simple Example

A simple example illustrates the long definition of HNFA+VM. Even though the learning algorithm is not yet presented, the resulting model taught with two dimensional toy data is discussed here. As is typical for nonlinear problems, the algorithm can only find a local optimum that depends on the initialisation. A local optimum of a good model is usually better than a global optimum of a bad one. In this case, the found network is not optimal but perhaps instructive. It is explained from bottom up.

HNFA+VM with three layers is taught with the two-dimensional data shown by dots in Figure 5.3. The data points correspond to different time indices t. The second layer, like the first one, has two dimensions: the first corresponds to the direction left and the second to the up right from the bias $\boldsymbol{\mu}_{s,1}$ marked with a circle. The single neuron in the third layer affects the neurons on the second layer and through them the data reconstructions. Possible values of the source $s_{3,1}$ form a curve in the data space.

Figure 5.4 illustrates the effects of the neuron in the third layer in more detail. At one end of its spectrum, it activates the first neuron in the second layer and at the other end, the second neuron. In the middle, both neurons are affected and the curvature of the data, is modelled. Most of the weights $\mathbf{B}$ to the variance neurons are close to zero, but the connection $B_{1,2,1}$ from the first dimension of the second layer to the vertical dimension of the data plane is very different. The effect can be seen in the lower part of the same figure.

Each data point is connected to the mean of its reconstruction $\mathbf{m}_1^s$ in Figure 5.5. The reconstructions are near the actual data points, except in the far left. In that region, the vertical variance neuron is activated. This explains why the reconstructions are allowed to be vertically inaccurate there.

The third layer has one source $s_{3,1}$ and the corresponding variance neuron $u_{3,1}$.

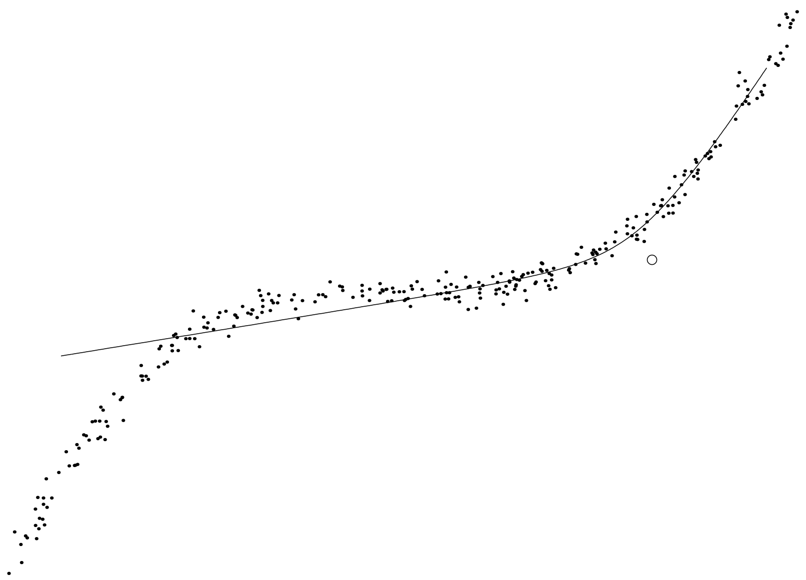
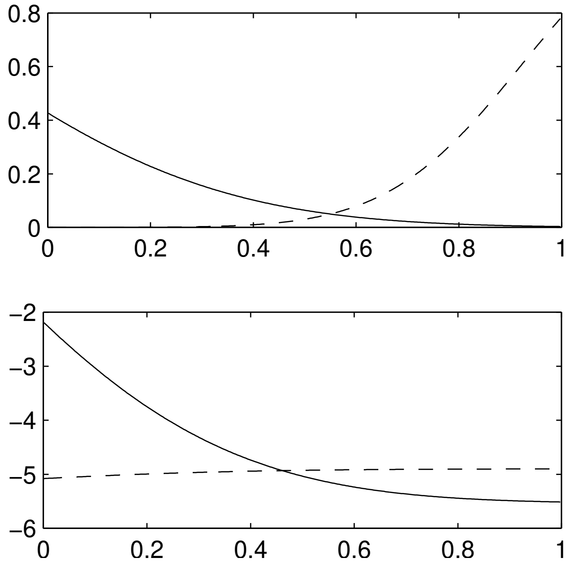


Figure 5.3: The data points $\mathbf{s}_1(t)$ of the simple example are marked with dots. The curve corresponding to different signals $s_{3,1}$ is shown. The bias $\boldsymbol{\mu}_{s,1}$ is marked with a circle.

The prior sdistribution of the source $s_{3,1}$ is depicted in Figure 5.6. The distribution is very close to a Gaussian, since the corresponding variance neuron happens to be inactive. Otherwise it would have been symmetric but super-Gaussian as explained in Section 2.3. The nonlinearity distorts the distribution such that the other end has a longer tail.

The behaviour of the model can be compared to that of a multilayer perceptron (MLP) network. In the MLP network, the sources of the second layer would have been computational hidden units and therefore their signals would have been functions of the third layer signal. In the HNFA+VM case the difference of input and output signals of the middle layer can be seen in Figure 5.7. The values are somewhat close to the diagonal line which means that the second layer behaves somewhat like a computational layer. The variation from the diagonal corresponds to the fact that the reconstructions in Figure 5.5 are not exactly on the curve. The variance neurons have no counterparts in the MLP framework.

With another initialisation the presumably globally optimal solution shown in Figure 5.8, was found. A third source in the second layer took care of the left part and all the reconstructions became relatively accurate. Variance neurons



$\mathbf{f}(\mathbf{s}_2)$ against $f(s_{3,1})$

$\mathbf{u}_1$ against $f(s_{3,1})$

Figure 5.4: The effects of the neuron in the third layer. Top: The signals $f(s_{2,1})$ (solid line) and $f(s_{2,2})$ (dash line) of the neurons in the second layer as the function of the signal $f(s_{3,1})$ of the neuron in the third layer. Bottom: The neuron affects also the variance neurons in the first layer. The vertical variance signal $u_{1,1}$ (solid line) and the horizontal variance signal $u_{1,2}$ (dash line) are plotted against $f(s_{3,1})$.

were of no use in this solution. The existence of local optimum in a simple problem like this suggests that it is far from trivial to design a good learning procedure and therefore the entire Chapter 6 is dedicated to it.

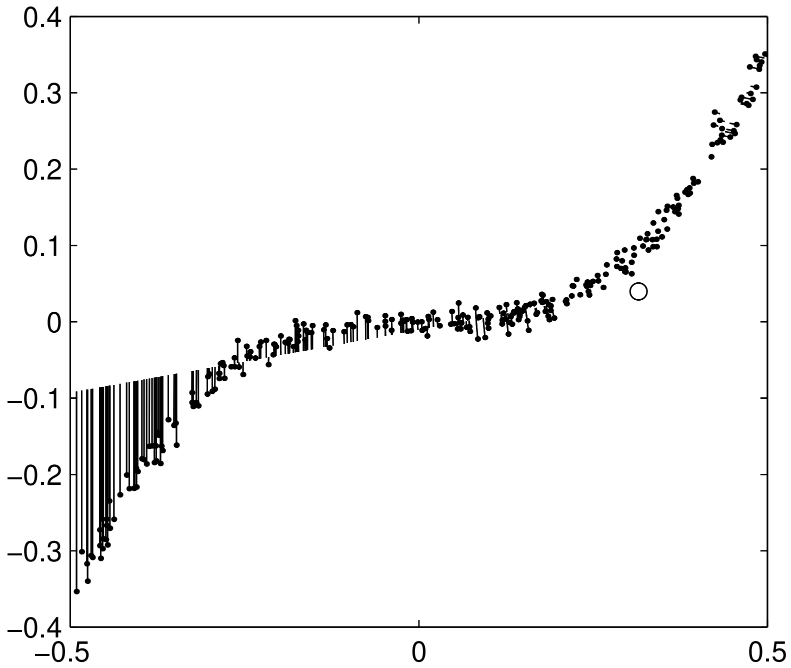


Figure 5.5: The data points $\mathbf{s}_1(t)$ marked with dots are connected to their reconstructions $\mathbf{m}_1^s(t)$ with a line. The bias $\boldsymbol{\mu}_{s,1}$ is marked with a circle.

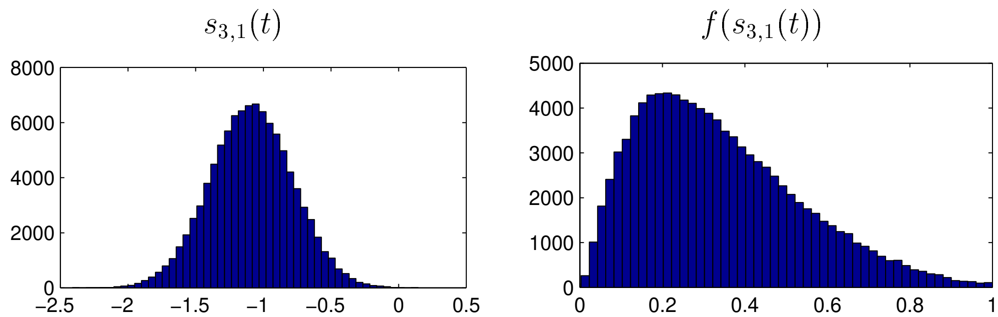


Figure 5.6: Left: The prior distribution of the neuron on the third layer. Right: The same distribution after the nonlinearity.

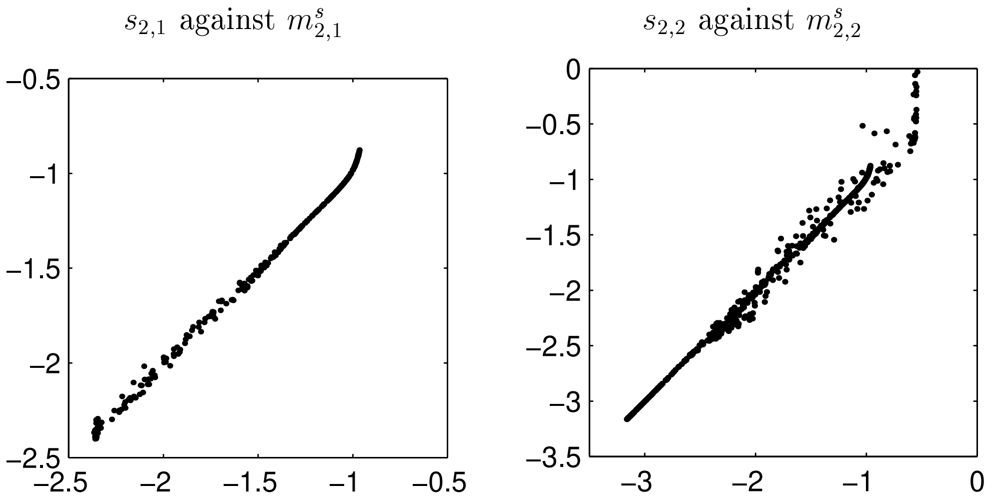


Figure 5.7: The posterior signals of source $\mathbf{s}_2$ are plotted against their prior signals $\mathbf{m}^s_2$ given by the upper layer. If the plots were exactly diagonal, the upper layer would define the signals in the second layer precisely.

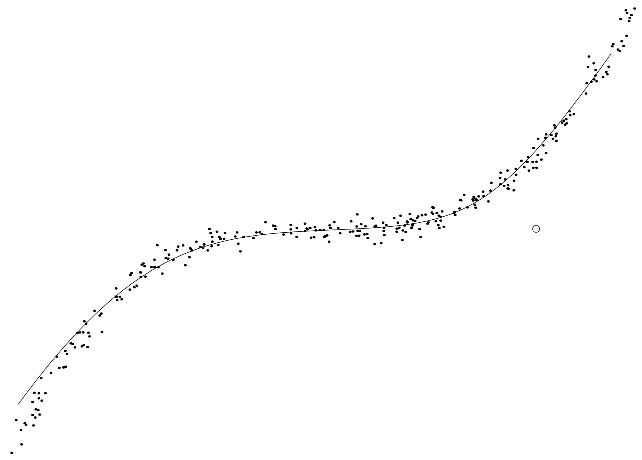


Figure 5.8: The presumably global optimum is visualised as in Figure 5.3.

Chapter 6

Learning Algorithm

This chapter describes the learning algorithm for the HNFA+VM model. The most important part of it, the adjustments of posterior approximations of single variables, is already described in Chapter 4, but as we already saw with the simple example in Section 5.3, there are many local minima in which the learning can get stuck. This makes the initialisation and tricks to avoid those minimax important.

The model structure defined in Chapter 5 is so rich that it can be used in various manners. It is possible that the sources on the uppermost layer define the reconstructions of the data quite exactly, that is, the neurons in middle layers are used like computational units. The second case would be that the an upper layer just activates the middle layer and the actual value is defined in the middle layer. The difference can be measured by separating the terms of the cost function to each layer. The first layer containing the observations can not be compared though, since it differs by not including the $C_{s,q}$ term defined in (4.2).

The initialisation in the first case should be quite different from the one used for the latter. If the reconstructions are defined already in the upper layer, it is unreasonable to initialise the model layer by layer. Instead, one could use the initialisation from [43]. The uppermost sources are initialised using PCA and the weights randomly. These sources are fixed for some period for the algorithm to find a meaningful representation of the data. It would be interesting to make such experiments with HNFA in the future.

The experiments in this thesis fall into the second category. The most important function of the upper layer is to activate some neurons in the middle layer and the actual reconstructions of the data are defined only in the middle layer. In this case, the weight matrices are initialised with vector quantisation or independent component analysis and fixed for some time to find meaningful values for sources.

6.1 Initialisation

The network is initialised as a single layer, that is $n = 1$. This means that there are only variance neurons connected to the observations. A new layer $i > 1$ can be added during the learning. The means of matrixes $\mathbf{A}_{i-1}$ and $\mathbf{B}_{i-1}$ are initialised by applying vector quantisation [1] to the whitened mean of concatenated vectors $\mathbf{s}_{i-1}(t)$ and $\mathbf{u}_{i-1}(t)$

$$\mathbf{x}_1(t) = \begin{pmatrix} \mathbf{s}_{i-1}(t) \\ \mathbf{u}_{i-1}(t) \end{pmatrix}. \tag{6.1}$$

The whitened vector $\mathbf{x}_2(t)$ of $\mathbf{x}_1(t)$ is obtained from singular value decomposition

$$\mathbf{x}_2(t) = \mathbf{D}^{-1/2}\mathbf{V}\mathbf{x}_1(t), \tag{6.2}$$

where $\mathbf{V}$ contains the orthonormal eigenvectors of the covariance matrix of $\mathbf{x}_1(t)$ and $\mathbf{D}$ is the diagonal matrix of its eigenvalues. Each $\mathbf{x}_2(t)$ is matched to one of the normalised model vectors $\mathbf{M}_k$

$$W(t) = \arg\max_k \mathbf{M}_k^T \mathbf{x}_2(t) \tag{6.3}$$

and the model vector is moved to the mean of the vectors $\mathbf{x}_2(t)$ that are matched to it

$$\mathbf{M}_k = \frac{\sum_{t|W(t)=k} \mathbf{x}_2(t)}{\sum_{t|W(t)=k} 1}. \tag{6.4}$$

Finally the initial values for $\mathbf{A}_{i-1}$ and $\mathbf{B}_{i-1}$ are

$$\begin{pmatrix} \mathbf{A}_{i-1} \\ \mathbf{B}_{i-1} \end{pmatrix} = \beta \mathbf{V}^T \mathbf{D}^{1/2} \mathbf{M}. \tag{6.5}$$

The scaling factor β should be selected such that the corresponding sources would operate in an appropriate range. Here the value $\beta = 2$ was used. It means that $f(s_i) = 1$ corresponds to twice the length of a model vector. The selection is further discussed in Chapter 9.

The posterior means of sources $\mathbf{s}_i(t)$ were initialised to -2 and the means of $\mathbf{u}_i(t)$ were initialised to -1. These very simple initial values of the sources are not harmful, because of a special state explained in Section 6.4.2. The posterior variances of $\mathbf{s}_i(t)$, $\mathbf{u}_i(t)$, $\mathbf{A}_{i-1}$ and $\mathbf{B}_{i-1}$ are initialised to small values.

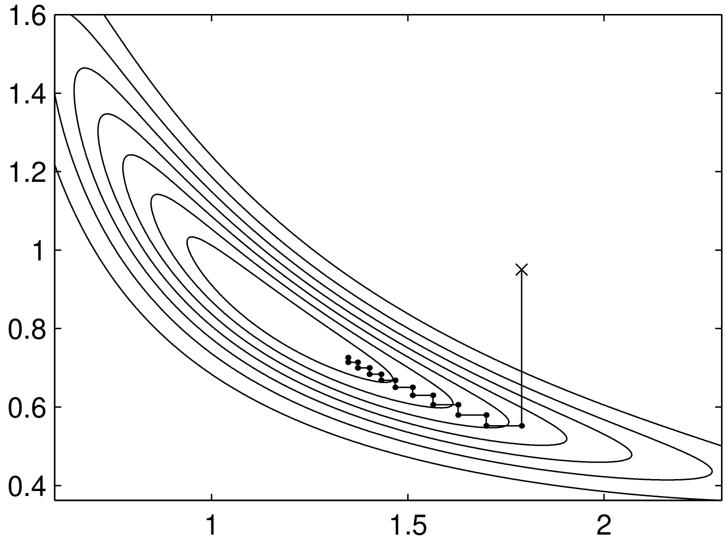


Figure 6.1: The posterior probability density of a simple problem: $x = s_1 s_2 + n, x = 1, p(s_k) = \mathrm{N}(s_k; 0, k), p(n) = \mathrm{N}(n; 0, 0.02)$. The density forms a ridge and the ascent for finding an MAP solution is not very effective, if the parametrisised distributions of s_1 and s_2 are updated alternately.

6.2 Adjustment

An essential part of the method is the iterative adjustment of the posterior approximation. Learning takes place when adjusting the network part by part using the update rules defined in Chapter 4. There are two implementations of the algorithm that differ at this point. In the Matlab version, vectors $\mathbf{s}_i$ and $\mathbf{u}_i$, matrices $\mathbf{A}_i$ and $\mathbf{B}_i$ and parameter vectors like $\boldsymbol{\mu}_{s,i}$ are updated one at a time keeping all other parts constant. In the C++-version every node is updated by itself keeping all other nodes constant. Updating many nodes at one time requires some further considerations [44]. The actual experiments were run using the Matlab version.

Alternating updating is the method that is used in this thesis, but it is not the only option. Figure 6.1 shows a simple example, where it is not very effective, since it leads to a zig-zag path. One option could be to sweep through many updates once and then optimise the length of the step in the direction of the whole sweep. This is further discussed in Chapter 9.

The alternating adjustment can be compared to the expectation maximisation (EM) algorithm [13]. The EM algorithm alternates between two types of adjustment steps in which the other part is kept constant.

6.2.1 Linear Computational Complexity

These updates consume most of the computing time of the algorithm. Therefore the computational complexity is of interest. As seen in Section 3.2, the cost function can be split up in a sum of simple terms. If each of them can be computed in constant time, the overall computational complexity is linear with respect to the number of connections.

In general, the computation time is constant if the parents are independent according to $q(\boldsymbol{\theta})$. The independence is violated if any variable receives inputs from a latent variable through multiple paths or from two latent variables which are dependent according to $q(\boldsymbol{\theta})$.

6.3 Learning the Structure

One of the reasons to use ensemble learning is the possibility to use the cost function for model selection or learning the structure (see Section 3.2.4). Modifications can be made to the structure and the cost function can be used directly to determine whether it was useful or not.

6.3.1 Pruning

Restricting the posterior approximation to have a factorial form effectively means neglecting the posterior dependences of variables. Taking into account posterior dependences usually increases computational complexity significantly. Often the computer time would be better used in a larger model with a simple posterior approximation. Moreover, often the latent variable models exhibit rotational and other invariances which ensemble learning can use by choosing a solution where the factorial approximation is most accurate (see [63] for an example).

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

The pruning tendency makes it easy to use for instance sparsely connected models because the learning algorithm automatically selects a small amount of well-determined parameters. In the early phases of learning, pruning can be harmful, however, because large parts of the model can get pruned away before a sensible representation has emerged. This corresponds to a local minimum of the algorithm. There are far less local minima with a posterior approximation taking into account the posterior dependences, but that would sacrifice computational efficiency. It seems that linear time learning algorithms cannot avoid local minima in general, but suitable choices of model structure and learning scheme can ameliorate the problem considerably.

If a neuron is effectively pruned away, it will not become useful again. In other words, is a local minimum with respect to the cost function for a neuron to be dead. If the neuron does not model anything, the output should be dampened off. If outputs are dampened off, it is not worthwhile to try to model anything. In practice, when a neuron is not useful, the weights that multiply its outputs diminish towards zero. For efficiency reasons these "dead" neurons are removed. They can be identified by estimating the cost function with and without them.

6.3.2 Regeneration

Neurons can also be added to an existing layer $i > 1$. Initialisation is similar to the case of a new layer except that the vector quantisation (or another method) is now applied to the reconstruction errors $\mathbf{s}_{i-1}(t) - \mathbf{m}^s_{i-1}(t)$ concatenated with $\mathbf{u}_{i-1}(t) - \mathbf{m}^u_{i-1}(t)$. The procedure of removing dead neurons and adding some new ones is called "regeneration". New neurons are in greater danger to be pruned away but Section 6.4.2 introduces ways to protect them.

6.4 Avoiding Nonglobal Minima

6.4.1 Rebooting

Some local minima can be avoided by sometimes resetting the neuron activities. It is called "rebooting". Time independent parameters and weights are left as they were but sources **s** and **u** are set to their means:

$$\forall i, k, t : \overline{s}_{i,k,\mathrm{new}}(t) = \frac{1}{T} \sum_{r=1}^{T} \overline{s}_{i,k}(r) \tag{6.6}$$

$$\forall i, k, t : \widetilde{s}_{i,k,\mathrm{new}}(t) = \frac{1}{T} \sum_{r=1}^{T} \widetilde{s}_{i,k}(r) \tag{6.7}$$

and similarly for **u**.

If there is lots of data from the same data set available, this is a good opportunity to change the data. It can be better to iterate more with less data than the other way around. Switching the data samples from time to time can still preserve some of the benefits of the larger data set. A network that performs well with new data, has a good generalisation capability by definition.

6.4.2 Special States

There are some special states in the learning algorithm. The states are layer-specific and can be active together. For example after rebooting, only the sources are updated for a while by keeping the weights constant. When learning an MLP-like network, the uppermost sources can be kept constant while updating the rest of the network. After adding a new layer to the network, the old parts can be kept constant and just learn the new part.

New neurons can be "kept alive" encouraging new neurons to be used instead of dampened off. This is achieved by modifying the propagated derivatives of the cost function in the multiplication node. The last term of derivate in equation (4.33),

$$2\mathrm{Var}\left\{s_2\right\} \frac{\partial C}{\partial \mathrm{Var}\left\{s_1 s_2\right\}} \left\langle s_1 \right\rangle \tag{6.8}$$

is ignored. It effectively means that the mean of s_1 is adjusted as if $\mathrm{Var}\left\{s_2\right\}$ were zero, in other words as if there were no uncertainty about s_2. In this way the cost function may increase at first due to overoptimistic adjustments, but it may pay off later on by escaping early pruning.

Chapter 7

Bars Problem

The first experimental problem studied with HNFA+VM was to apply it to artificial data that is an extension of the bars problem [12]. The data set consists of 1000 image patches of size 6×6 pixels. They have horizontal and vertical bars. In addition to the regular bars, the problem was extended to include horizontal and vertical variance bars that are manifested by increased variance. Samples of the image patches are shown in Figure 7.1.

Data was generated by first choosing whether vertical and/or horizontal orientations are active, each with probability 1/2 independently. When an orientation is active, there is a probability 1/3 for each bar to be active. For both orientations, there are 6 regular bars, one for each row or column, and 3 variance bars that are 2 rows or columns wide. The intensities are drawn from normalised positive exponential distribution. Regular bars are additive and variance bars produce additive Gaussian noise with standard deviation of its intensity. Finally, Gaussian noise with standard deviation 0.1 is added to each pixel.

7.1 Learning Procedure

The network was built up in stages shown in Figure 5.2. It was initialised with only a single layer. The second layer was created at sweep 20 and the third at 100. After creating a layer only its sources were updated for 10 sweeps and it was "kept alive" for 50 sweeps. The system was "regenerated" at sweeps 300 and 400 and after that only the sources were updated for 5 sweeps and the system was "kept alive" for 50 sweeps. The network was "rebooted" at sweeps

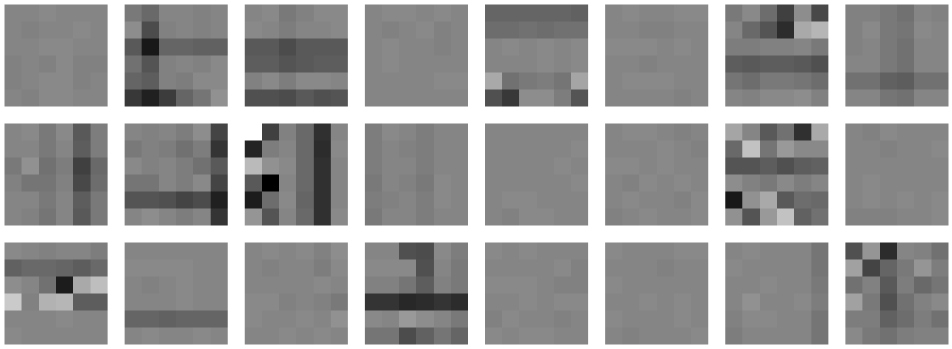


Figure 7.1: Samples from the 1000 image patches used in the bars problem.

500, 600 and 700 and only the sources were updated for the next 40 sweeps. Each sweep corresponds to going through all the data vectors and updating each latent variable node once.

The size of the first layer n_1 is equal to the size of the data vector, that is 36. The second layer was created with 30 neurons and additional 3 in each regeneration. The third layer layer was created with 5 dimensions and additional 2 in regeneration. Dead neurons were removed every 20 sweeps.

This learning procedure was designed for this problem. Future work includes the complete automation of the learning process including the selection of the number of sweeps.

7.2 Results

The initial weights of the first layer can be seen in Figure 7.2. Some regular bars are visible, but there are multiple bars in the same patch. Variance bars are even less clear. There are a few sources that diminish variance, which means that they are active when neither of the orientations were in the generation process.

Figure 7.3 shows the weights after adding another layer. Regular bars are quite well formed in $\mathbf{A}_1$. Two sources represent the same rightmost vertical bar, though. The upper horizontal variance bar and the right vertical variance bar are somewhat mixed up in $\mathbf{B}_1$. The leftmost source on the second row clearly diminishes variance. The weights $\mathbf{A}_2$ and $\mathbf{B}_2$ of the upper layer seem to be quite useless at this stage.

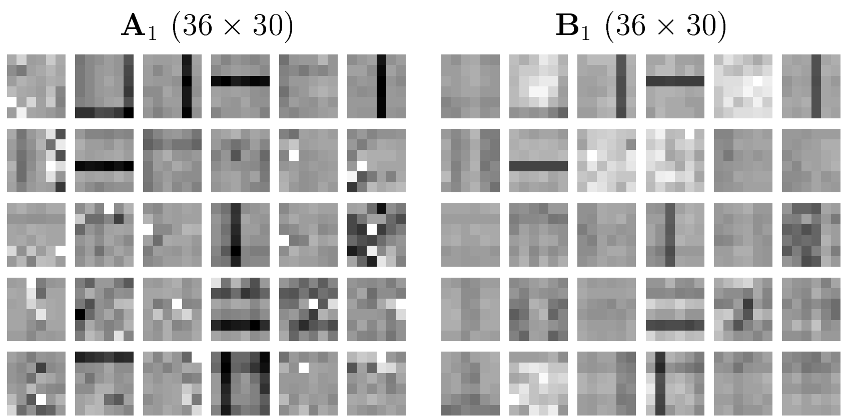


Figure 7.2: Posterior means of the weight matrices after 20 sweeps. The second layer has just been initialised. The matrices are organised in patches and dark shades represent positive values.

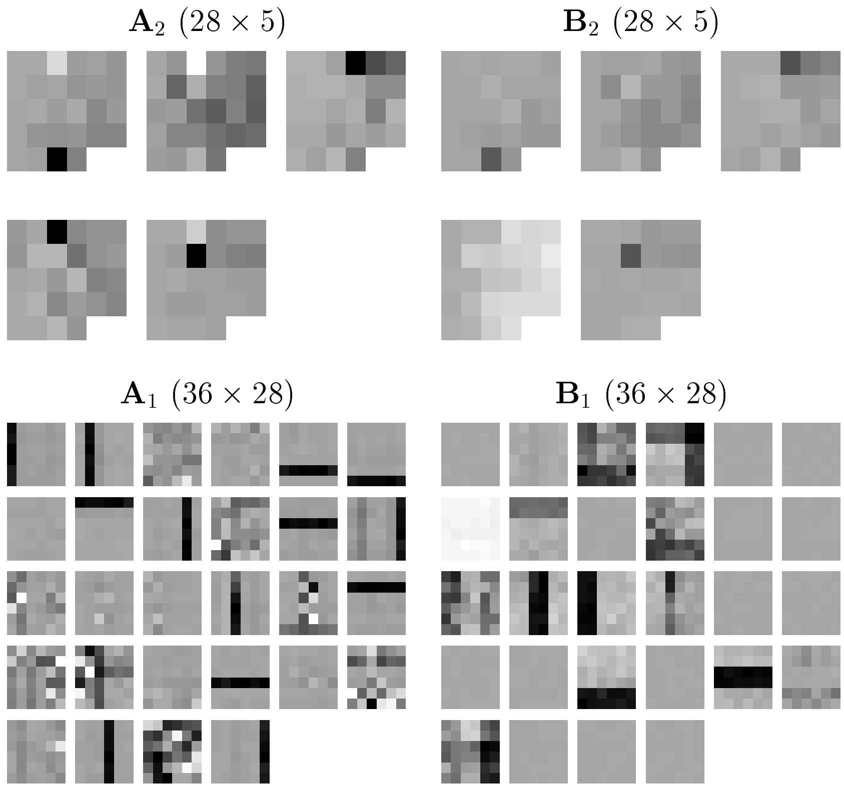


Figure 7.3: Posterior means of the weight matrices after 100 sweeps. The third layer has just been initialised. The five patches in $\mathbf{A}_2$ and $\mathbf{B}_2$ correspond to the five sources on the third layer. The pixels of $\mathbf{A}_2$ and $\mathbf{B}_2$ correspond to the 18 sources on the second layer and thus to the patches of $\mathbf{A}_1$ and $\mathbf{B}_1$.

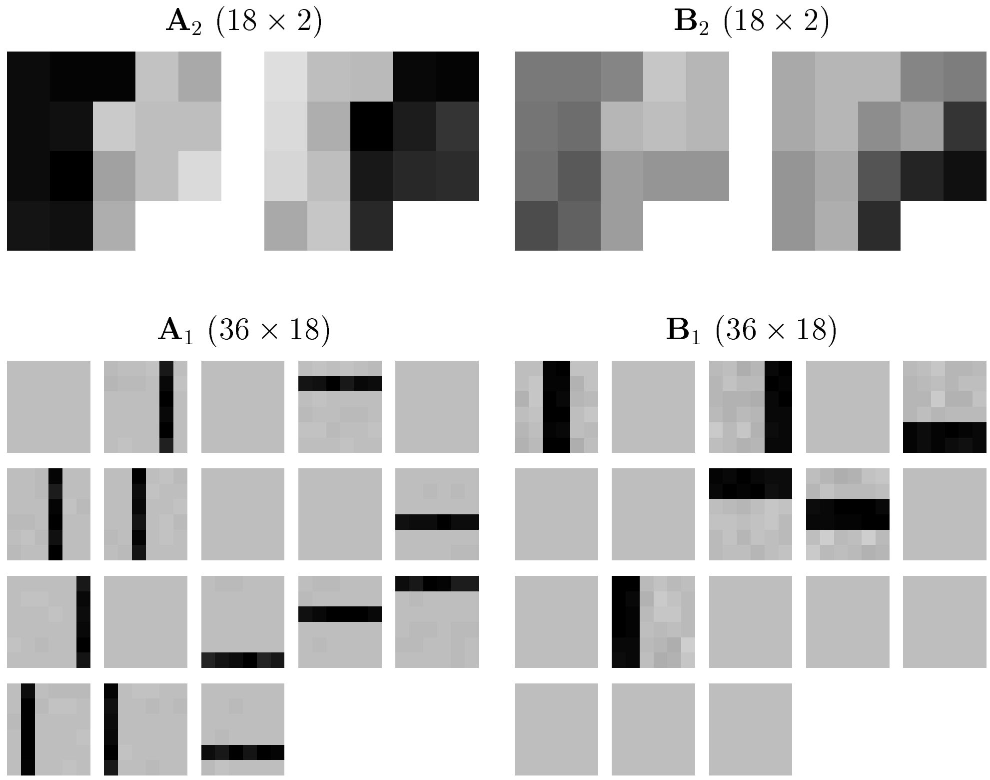


Figure 7.4: Final results: Posterior means of the weight matrices after 1200 sweeps. The sources of the second layer are ordered for visualisation purposes according to the weights $\mathbf{A}_2$ and $\mathbf{B}_2$ using self-organising map.

Figure 7.4 demonstrates that the algorithm finds a generative model, that is quite similar to the generation process. The two sources on the third layer correspond to the horizontal and vertical orientations and the 18 sources on the second layer correspond to the bars. Regular bars, present in $\mathbf{A}_1$, are reconstructed accurately but the variance bars in $\mathbf{B}_1$ exhibit some noise. The distinction between horizontal and vertical orientations is clearly visible in $\mathbf{A}_2$.

Figure 7.5 shows the mixing matrices from principal component analysis (PCA) and independent component analysis (ICA) with the same data. It should be noted that in PCA and in ICA, there is a symmetry between dark and light shades and it is irrelevant, which one is shown. These models are not rich enough to find variance bars. The regular bars, however, are found to a degree. In PCA, the bars are mixed up especially with other bars of the same orientation. There should be 12 bar patches, but the last ones are crippled with noise. ICA does not mix the bars with each other so much. There are no

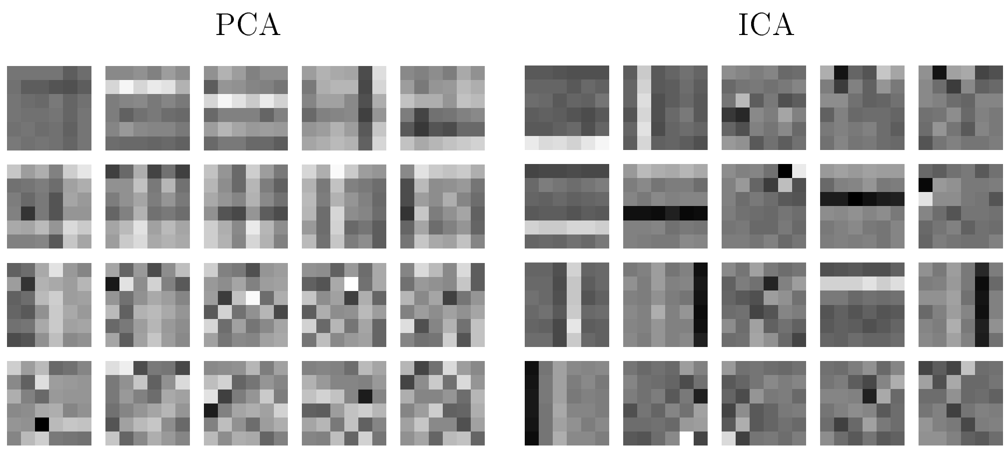


Figure 7.5: The mixing matrices found by PCA and ICA algorithms.

mixtures of horizontal and vertical bars. Only ten patches show bars which means that some must be missing. Results with vector quantisation (VQ) are practically the ones used in the initialisation shown in Figure 7.2. Some bars are found well, but some are missing and some patches are mixtures of horizontal and vertical bars.

Chapter 8

Experiments with Image Data

HNFA+VM, presented in Chapter 5, was tested with a number of natural gray-scale images as a data set. Gaussian noise with standard deviation 0.1 was added to the images to avoid artefacts caused by the discrete gray levels from 0 to 255. The intensities were scaled to variance one.

10 times 10 image patches were taken randomly from the images to be used as data vectors. There was a total of 10000 data vectors. The data matrix $\mathbf{X}$ is thus 100 by 10000. The mean of each patch was subtracted from the patch and the data was whitened to a degree $\alpha = 0.8$ and rotated back to the original space:

$$\mathbf{X}_{new} = \mathbf{V}^T \mathbf{D}^{-0.5\alpha} \mathbf{V}\mathbf{X}, \tag{8.1}$$

where $\mathbf{V}$ contains the orthonormal eigenvectors of the covariance matrix of the data and $\mathbf{D}$ is the diagonal matrix of its eigenvalues. Regular whitening corresponds to $\alpha = 1$ and whitening to a degree 0 would leave the data unchanged. The partly whitened version of the data was rotated back to the original space by multiplying with $\mathbf{V}^T$ from the left so that the dimensions of the data would still correspond to the pixels. Whitening is used, because the dominating feature of the images is the positive correlation between nearby pixels and the model could otherwise spend a layer just to model that. Regular whitening is typically used as a preprocessing for ICA.

Figure 8.1 shows the matrix $\mathbf{V}$ or the principal components of the data. There are only 99 components, since the removal of the mean in each image removes also one of the intrinsic dimensions. There is a great resemblance to the discrete cosine transform (DCT), which is widely used in image compression [21]. Compression and ensemble learning have much in common as was seen in Subsection 3.2.3. Taking into account that there are efficient algorithms for

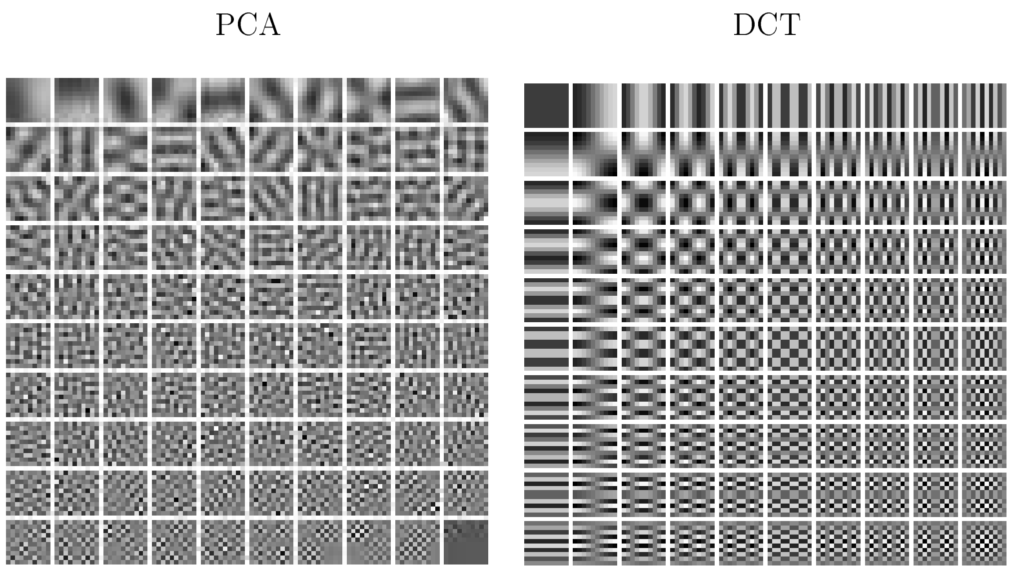


Figure 8.1: Left: The mixing matrix resulting from applying Principal Component Analysis (PCA) to the same data set. Right: the basis of the Discrete Cosine Transform (DCT) that is used in image compression.

calculating the DCT, it is clearly a good choice for compression. None of the patches are localised in either PCA or DCT.

8.1 Learning Procedure

The phases of the learning procedure are explained in Chapter 6. The initialisation of $\overline{\mathbf{A}_1}$ was done with FastICA algorithm [28]. The initialisation procedure is quite similar to the one with VQ. The results with ICA are presented here, since they were somewhat better than the ones using VQ. Future work should include a more careful comparison of different initialisation methods. As ICA is symmetric with respect to positive and negative values, the mixing matrix was doubled to include the negative version as can be seen in Figure 8.2. The sources were updated for 100 sweeps. Then the reconstruction error was fed to ICA again, now including also the variance sources as described in Subsection 6.3.2. It results in some more neurons on the second layer which can be seen in Figure 8.4.

The sources were updated for one hundred sweeps and the least useful ones were removed. Now the second layer had 210 neurons. The sources were updated until sweep 500 when the sources $\mathbf{s}_2$ and variance sources $\mathbf{u}_2$ of the

$\mathbf{A}_1$ from ICA (100×140)

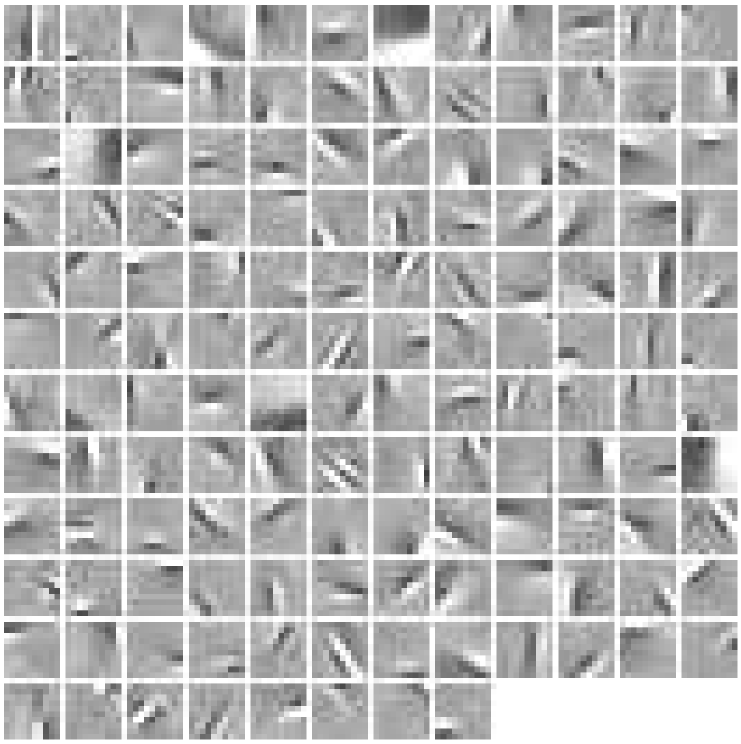


Figure 8.2: 70 independent components and the same as negative versions are used as the initialisation of $\overline{\mathbf{A}_1}$. $\overline{\mathbf{B}_1}$ is initialised to zero.

second layer were fed to ICA once again to get initial values for $\mathbf{A}_2$ and $\mathbf{B}_2$ in Figure 8.5. The new sources on the third layer were updated for 200 sweeps and during that the second layer sources were updated every fifth sweep.

The sources on the second layer are ordered for visualisation purposes based on the connections from the third layer. Each dimension of the means of the weights $\mathbf{A}_2$ and $\mathbf{B}_2$ are scaled to zero mean and unit variance and fed to the self-organising map (SOM) [39]. The patches are then organised close to their best matching unit in the SOM.

Next, also the weights were released to be updated. The second layer was "kept alive" for 1500 sweeps. Figure 8.6 shows the situation at sweep 1000. The algorithm has simplified the model by killing neurons. The "dead" neurons are removed and everything is updated without using the special states until finally at sweep 6000 the final results are shown in Figure 8.7.

$\mathbf{A}_1$ from VQ (100×200)

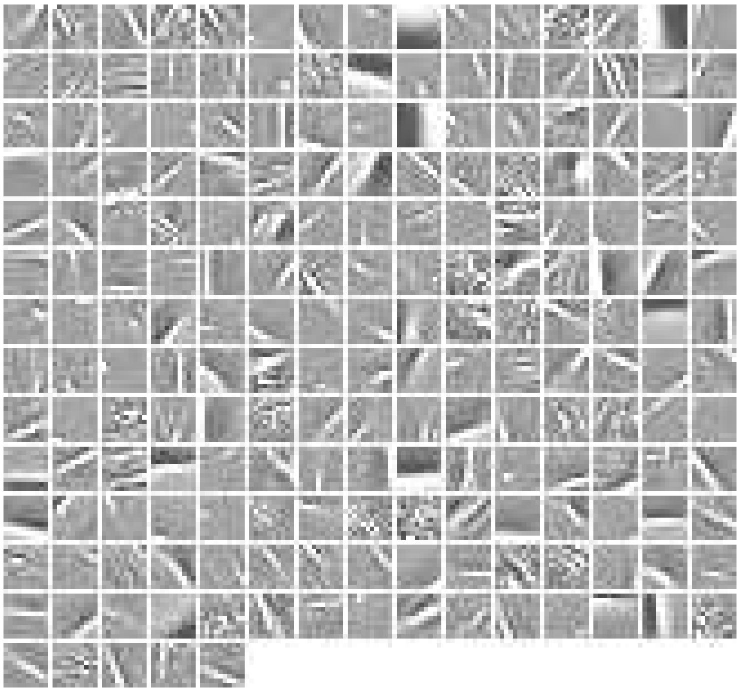


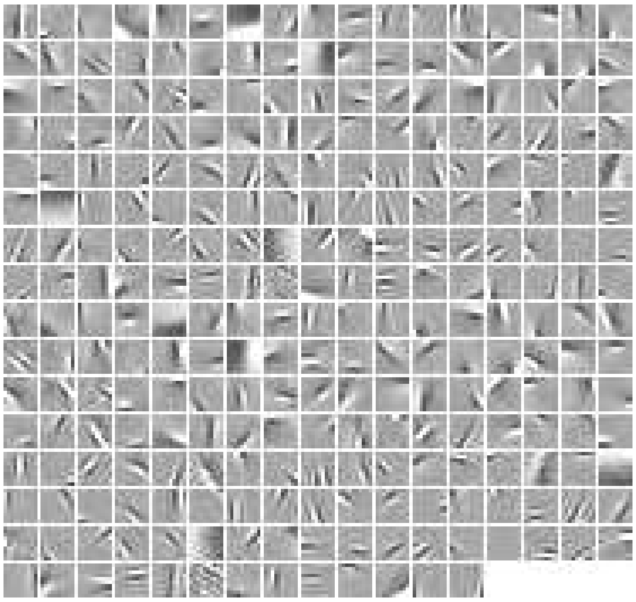
Figure 8.3: The vector quantisation (VQ) initialisation with 200 model vectors is shown for comparison. There are more vectors that use the whole area since VQ yields a maximally sparse representation. VQ does not have the symmetry between light and dark and light features are clearly dominating.

8.2 Results

The resulting weights are shown in Figure 8.7. The neurons in the upper right corner of $\mathbf{A}_1$ are specialised to just a few data samples. Two neurons in the lower left corner have connection in the $\mathbf{B}_1$ matrix thus adding only noise to the reconstruction and three others in both $\mathbf{A}_1$ and $\mathbf{B}_1$. Other $\mathbf{B}$ connections are close to zero. Reading from upper left corner in $\mathbf{A}_2$ the first neuron of the third layer activates the specialised neurons and inhibits the neurons that add noise. Second and fourth neuron activate noise neurons and inhibits the low frequency neurons in the lower right of $\mathbf{A}_1$. The third neuron activates neurons with diagonal features from upper left to lower right corner and the fifth neuron activates the horizontal features.

The initialisation seems to affect the results considerably. This was already noticed with the simple example in Section 5.3. The algorithm can get stuck in

$\mathbf{A}_1$ (100 × 268)



$\mathbf{B}_1$ (100 × 268)

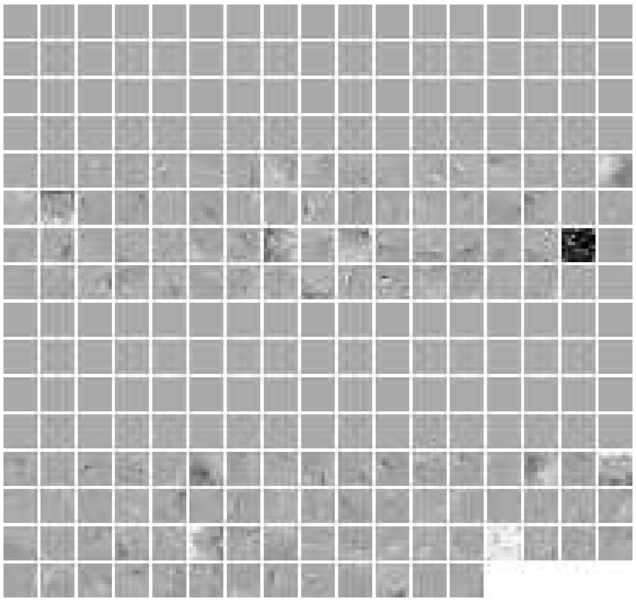


Figure 8.4: After 100 sweeps through the data. The reconstruction error of $\mathbf{s}_1$ and $\mathbf{u}_1$ are used as data for another run on ICA to get more basis vectors. Now the matrix $\overline{\mathbf{B}_1}$ has nonzero entries, too.

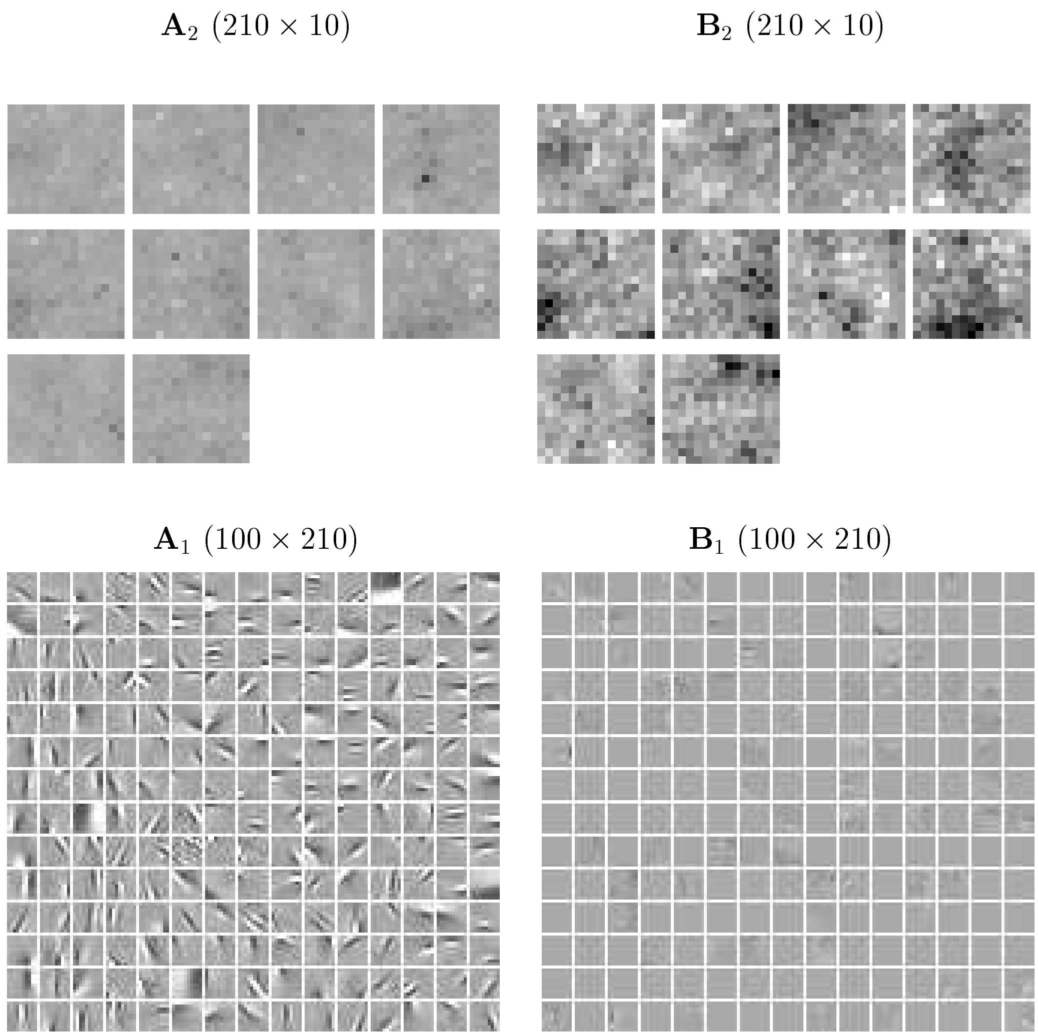


Figure 8.5: After 500 sweeps through the data. The least useful sources have been pruned away. The sources of the second layer $\mathbf{s}_2$ and $\mathbf{u}_2$ are fed to ICA to get initial values for $\mathbf{A}_2$ and $\mathbf{B}_2$. The sources have been ordered based on $\mathbf{A}_2$ and $\mathbf{B}_2$. The ten patches in $\mathbf{A}_2$ and $\mathbf{B}_2$ correspond to the ten sources on the third layer. The pixels of $\mathbf{A}_2$ and $\mathbf{B}_2$ correspond to the 210 sources on the second layer and thus to the patches of $\mathbf{A}_1$ and $\mathbf{B}_1$.

a local minimum of the cost function and even regeneration of the neurons does not always help, since when there already is an adequate reconstruction of the data, it is very hard for a new neuron to fit in. The asymmetric initialisation with VQ seems to result in more asymmetric features than the ones presented here. Some more experiments need to be done to confirm that. I would like to stress that a local optimum of a good model is usually better than a global

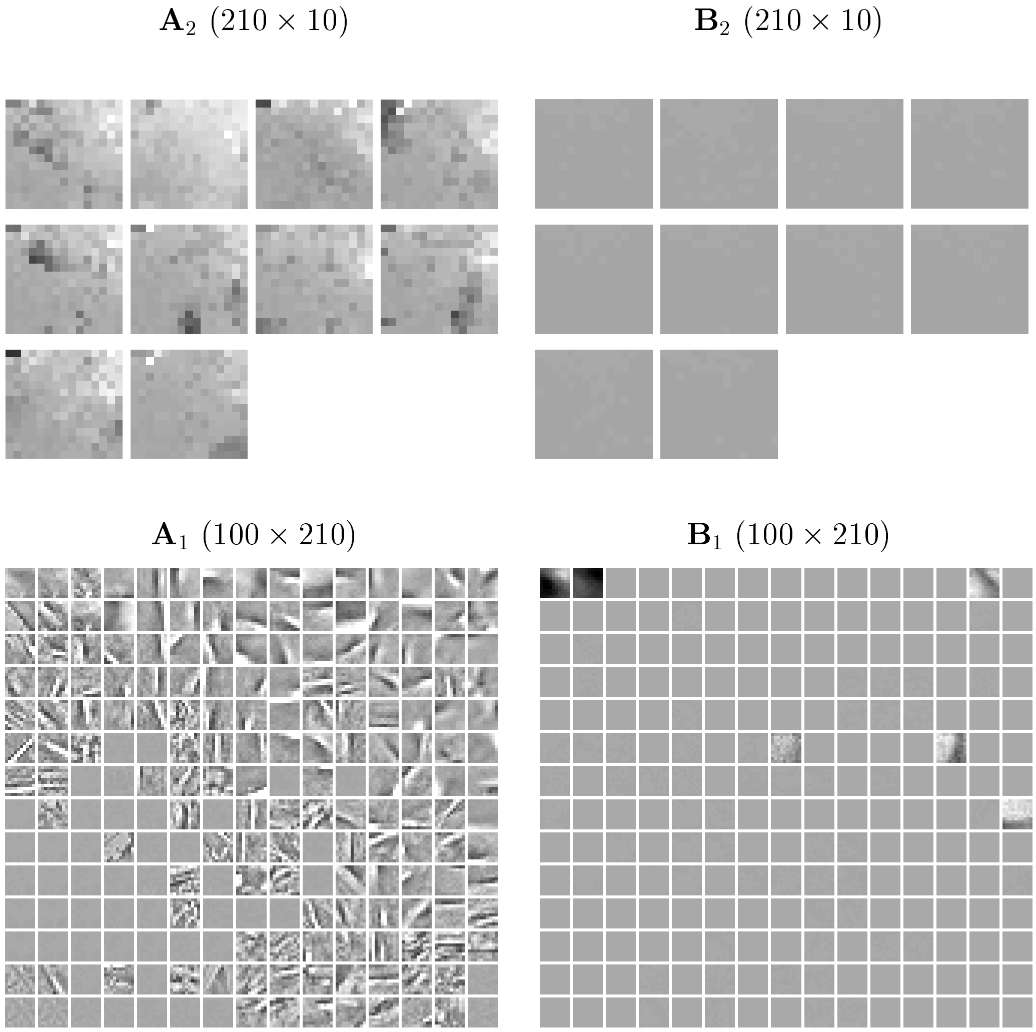


Figure 8.6: At sweep 1000. After releasing the weights the algorithm has started to simplify the model. Dead neurons can be seen in the lower left corner of $\mathbf{A}_1$. The patches in $\mathbf{B}_1$ are either clearly close to zero or clearly differ from zero. The emphasis on the second layer has moved from $\mathbf{B}_2$ to $\mathbf{A}_2$.

optimum of a bad one.

In the simple example in Section 5.3, the terms of the cost function corresponding to the third layer were greater than the ones corresponding to the second layer. This means that the reconstructions are effectively decided on the third layer and just focused on the second layer or that the neurons on the second layer are acting like computational units. But in this experiment the proportion is one in the third layer against eight in the second layer. This means that the reconstruction of the image is decided on the second layer and

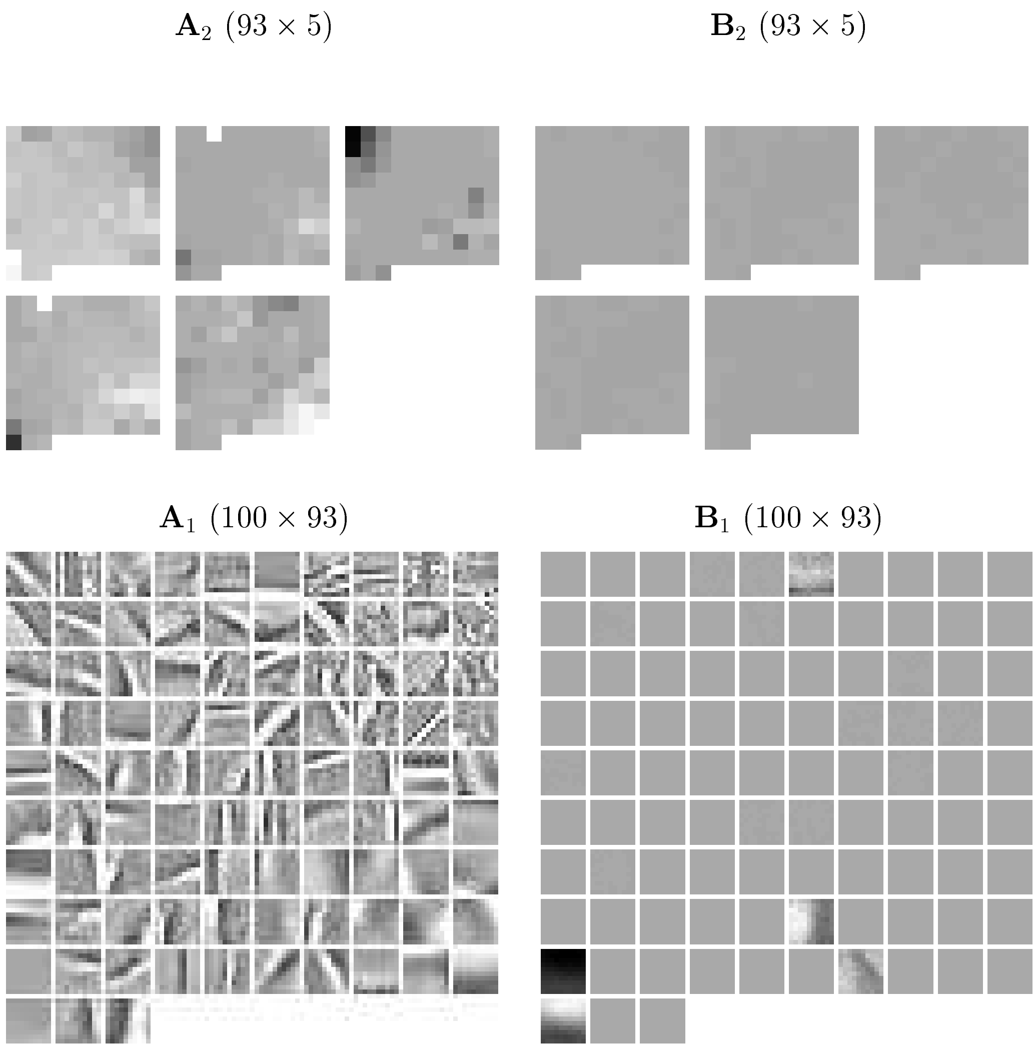


Figure 8.7: Results at sweep 6000. "Dead" neurons have been removed.

the third layer only guides it.

At the end of the iteration process the cost function would have become smaller by removing some neurons that were stuck in local minima. After the network is rebooted and updated some more, the number of neurons alive still diminishes. As there are sources on the third layer that correspond to horizontal and diagonal features in the second layer, there should also be sources corresponding to the vertical and the other diagonal directions. There is work to be done to improve the results.

The resulting patches in $\mathbf{A}_1$ resemble the wavelets or features found by ICA. They are more localised than features of the PCA but only the features that fill most of the patch seem to survive. They seem to follow the prior that is set to them in Section 5.2.2. Each patch is either close to zero or active in the whole patch or a large part of it. This would suggest that a sparse prior for the weights could help to get more local features like the ones from ICA algorithm in Figure 8.2. Preliminary experiments do support this assumption. It would also help to keep more neurons alive and make the reconstructions more accurate. Using a larger set of data would also help by making the relative cost of describing the weights smaller.

Sparse connectivity would also help the upper layer. As the number of features grows, there is a growing amount of cost to describe that a particular neuron in the third layer does not affect activities of most of the neurons on the second layer. With sparse connections the situation would be very different. It would be useful to have a neuron on the third layer that states that just two of the features are typically active together. When increasing the size of the image patches, this locality would become increasingly important.

The assumptions made in the posterior approximation might explain why either $\mathbf{A}$ or $\mathbf{B}$ describing the connections to a particular neuron and its variance neuron tend to get turned off. Upper layer cannot be connected straigth to the mean and variance prior of a neuron, because it would violate the independency assumption. The addition of the variance neuron in between restores the independency, but taking the dependency not into account increases the cost. The increase is smallest, when the dependency is smallest. If the connections $\mathbf{A}$ and $\mathbf{B}$ were similar to each other, not only would the wrong assumption of the independence rise the cost function, but also the cost would include the description of the weights, which are also assumed to be independent of each other.

The experiments on image data using topographic ICA and independent subspace analysis [30, 29, 31] are perhaps closest to these ones. The features found with these methods corresponding to $\mathbf{A}_1$ are similar to those of the basic ICA. The topography or the collection of subspaces correspond approximately to a fixed, predetermined matrix $\mathbf{B}_2$. In case of the topology, there would be a Gaussian spot in each patch of $\mathbf{B}_2$. In case of independent subspace analysis, each patch of $\mathbf{B}_2$ would activate a distinct group of features or the independent subspace. These methods do not have corresponding parts for $\mathbf{B}_1$ and $\mathbf{A}_2$. It seems that ICA has not previously been used to learn all parts of hierarchical models succesfully.

Chapter 9

Discussion

The aim of this work was to build a model which could be used to find out statistical features in data in an unsupervised manner. Many of the previously presented methods scale poorly with respect to the number of intrinsic dimensions in the data or cannot take into account nonlinear effects. A model called hierarchical nonlinear factor analysis with variance modelling (HNFA+VM) is proposed.

The commonly used maximum likelihood (ML) and maximum a posteriori (MAP) learning criteria suffer from overfitting and thus cannot be used with nonlinear factor analysis models. A Bayesian approach based on sampling would be too slow for large problems. Ensemble learning is a good compromise between them and was therefore selected.

The complicated model is built from simple blocks which reduces the effort of implementation and increases extensibility. All computations are local which results in linear computational complexity. The idea for the structure of the model originates from nonlinear factor analysis (NFA) [43]. The computational or hidden units are replaced by latent variables. The same blocks can also be used to model variances which has been found important for analysis of image data [67, 30].

The proposed learning algorithm for HNFA+VM was shown to be able to learn the structure of the underlying artificial but complicated process that generated the data. The experiments with real world data are not yet very convincing but it seems that introducing sparse connectivity could enhance the results significantly.

The computational complexity of the learning algorithm for HNFA+VM is linear with respect to the number of connections, sample vectors and sweeps. Still, the experiment with image data took more than a week to run with the Matlab version. A more efficient C++ version of the algorithm has become ready while writing this. It is also more flexible for example by allowing the pruning of single connections instead of whole neurons.

A sparse prior for the weights is likely to prove useful in many cases. On the first layer of the image experiments it would encourage local features to be formed in a patch. On the upper layer it would encourage the formation of complex-cell-like [30, 29] sources as each of them typically models the variance of only a small number of the lower-level sources. The connections that are practically zero can be pruned away to make the algorithm significantly more efficient.

The number of time indices or observations is also crucial to the efficiency. When looking at a picture or a view, the human eye focuses in places of interests and not randomly. Perhaps a smaller set of data would suffice for interesting results if the data set would be selected in the same manner. Also, the rebooting and changing the data that is used should be studied more. There might be ways to take into account that the data has been changed. In case of online learning, one can use new data all the time. It is easy to use online learning with Bayesian methods. Preliminaly results with it are promising.

The learning with alternating adjustment is inefficient in cases where different parts can compensate each other. For finding the best rotation matrix, one should be able to adjust several parts of the model at once. To use a different part of the nonlinear function to get a different curvature, one should be able to compensate the affected scaling and bias terms at the same time. A resulting zig-zag path illustrated in Figure 6.1 leads to the right direction but very slowly. One could identify internally related groups of parameters whose adjustments are steady between sweeps and then make a one-dimensional optimisation along the direction of these steady updates. This could dramatically decrease the number of required sweeps. The system could also be made to predict which kinds of updates are the most effective ones.

The building blocks discussed in this thesis together with two more blocks presented in [66] can be used to build a wide variety of models. An important future line of research will be the automated construction of the model. The search through different model structures is facilitated by the ability of ensemble learning to automatically shut down parts of the model. The cost function can be used to determine wheather a modification of the structure is useful or not. The rate of decrease of the cost can be used to estimate the utility of further sweeps.

The building blocks can be connected to other time instances thus modelling the nonlinear dynamics of the sources. Valpola got good results [65, 63] using nonlinear factor analysis with another nonlinear mapping from sources at time t to sources at time $t+1$. With image data, this would mean that in an animation the higher level sources would change slowly. This would encourage them to represent features that are for example translationally more or less invariant since an animation typically looks like a translation locally. This is promising, since some of the higher level sources did activate for example different horizontal edge features even with the static images.

Externally the variance neurons appear as any other Gaussian nodes. It is therefore easy to build also dynamic models for the variance. These kinds of models can be expected to be useful in many domains. For example volatility in financial markets is known to have temporal auto-correlations.

The scope of this thesis was restricted to models with purely local computations. In some cases it may be necessary to use models where a group of simple elements is treated as a single element whose external computations are local but whose internal computations may be more complex. The practical consequences of using a latent variable instead of a computational node in NFA should be studied.

The benefits of the proposed method are as follows. First, the method is unsupervised, that is, the learning does not require a teacher or human intervention. Second, the proposed structure is quite rich. It is not restricted to linear manifolds, it includes the modelling of the variance and the number of layers in the hierarchy is not restricted. Third, the computational complexity of the algorithm scales linearly with respect to the size of the problem, which means that the scalability is very good. Fourth, the method avoids overfitting which leads to good generalisation capability and requires no cross-validation. Finally, different model structures can be compared simply by using the cost function.

The restrictions of the method at its current state include the requirement of expertise in selecting preprocessing, initialisation, and the learning procedure since it is prone to local minima. The algorithm is computationally intensive compared to other simpler methods. A universal data analysis method should also be able to use data in other forms than just continuous valued vectors with constant number of dimensions. At least discrete values and relations to other observations would be useful. For example, real medical databases contain a mixture of measurements, natural language, images and relations to

	M		M		M	
M	M	M	M	M	M	M
	M		M		M	
M	M	M	M	M	M	M
	M		M		M	
M	M	M	M	M	M	M
	M		M		M	

Figure 9.1: A statistical model of natural images can be used to form a super-resolution image. It can be done by considering the pixels marked with "M" missing and reconstructed by the model.

hospitals, doctors and treatments.

The assumption that the data is continuous valued is seldom exactly true. Even if the underlying phenomenon is continuous, the data might be more or less discrete. Prices tend to be rounded. Ages are typically given in full years. Digital images are typically discretised to for example 256 gray-scale values. These artefacts can cause unwanted phenomena in learning. They can be avoided by adding a small amount of random noise to the data set as was done in the experiments here. It would also be possible to add the noise implicitly by giving a virtual posterior variance for the observed samples. This idea is left for further studies.

The proposed model can be used to statistical analysis of data. It can be used to reconstruct missing values and thus to make predictions. The found sources can be used as features for other machine learning methods. The method can be further developed to be an artificial intelligence system. A concrete example application with images is visualised in Figure 9.1. Superresolution images can be obtained by adding extra pixels in between the original ones and considering them as missing values. The model could then reconstruct them in a manner explained in [57].

Real applications with image data require the use of images much larger than 10×10 pixels. Even if the algorithm scales linearly, it would be reasonable to explicitly take into account the translational invariance in images. One could start the learning process with smaller patches and use the results as an initialisation for learning with larger patches. The computationally expensive lowermost layers would be thus learned with smaller patches and the upper layers could combine the local features of them later on.

The effect of the initialisation on the final results should be studied. It seems that different kinds of initialisations result in different kinds of models. Therefore different methods for initialisation should be tested for different kinds of data. One of the methods that could be good for initialisation, is the nonlinear component analysis (NCA) [60], which would differ from the methods used so far by being nonlinear.

Scaling of the model vectors in the initialisation requires some more attention. By comparing the initial scales to the resulting model, one can find a more appropriate scaling factor β which could later be used for a better initialisation and thus faster convergence. Another option would be to project the data to a model vector and compare the resulting distribution to one given by a nonlinear unit. A good scaling factor would be one that matches the distributions best.

The C++-version of the algorithm allows the connections to be made more freely. One could connect the upper layers directly to the lower layers in addition to using layers in between. This could decrease the effect of a particular initialisation method.

Bibliography

[1] H. Abut, editor. *Vector Quantization.* IEEE Press, New York, 1990.

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

[3] D. Barber and C. M. Bishop. Ensemble learning for multi-layer networks. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10, NIPS*97*, pages 395–401, Denver, Colorado, USA, Dec. 1–6, 1997, 1998. The MIT Press.

[4] E. Beale and C. Mallows. Scale mixing of symmetric distributions with zero means. In *Annals of Mathematical Statistics*, 1959.

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

[6] C. M. Bishop, M. Svensén, and C. K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998.

[7] C. M. Bishop and J. M. Winn. Non-linear Bayesian image modelling. In D. Vernon, editor, *Proceedings of the 6th European Conference on Computer Vision, Part I. Lecture Notes in Computer Science*, pages 3–17, Dublin, Ireland, 2000. Springer.

[8] J.-F. Cardoso. Multidimensional independent component analysis. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, ICASSP'98*, pages 1941–1944, Seattle, Washington, USA, May 12–15, 1998.

[9] C. Chen, editor. *Neural Networks For Pattern Recognition And Their Applications.* World Scientific, Singapore.

[10] R. Choudrey, W. Penny, and S. Roberts. An ensemble learning approach to independent component analysis. In *Proceedings of the IEEE workshop on Neural Networks for Signal Processing*, Sydney, Australia, 2000.

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

[12] P. Dayan and R. S. Zemel. Competition and multiple cause models. *Neural Computation*, 7(3):565–579, 1995.

[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)*, 39:1–38, 1977.

[14] P. Földiak and M. P. Young. Sparse coding in the primate cortex. In *The Handbook of Brain Theory and Neural Networks*, pages 895–898, Cambridge, Massachusetts, 1995. The MIT Press.

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

[16] B. Frey and N. Jojic. Estimating mixture models of images and inferring spatial transformations using the EM algorithm. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 416–422, 1999.

[17] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. In *Neural Networks, vol. 2*, pages 183–192, 1989.

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

[19] Z. Ghahramani and G. E. Hinton. Hierarchical non-linear factor analysis and topographic maps. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10, NIPS*97*, pages 486–492, Denver, Colorado, USA, Dec. 1–6, 1997, 1998. The MIT Press.

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

[21] R. C. Gonzales and R. E. Woods. *Digital Image Processing.* Addison-Wesley, 3rd edition, 1992.

[22] H. Harman. *Modern Factor Analysis.* University of Chicago Press, 2nd edition, 1967.

[23] T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84:502–516, 1989.

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

[25] G. E. Hinton and D. van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 5–13, Santa Cruz, California, USA, July 26–28, 1993.

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

[27] D. Hubel and T. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology of London*, 160:106–154, 1962.

[28] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.

[29] A. Hyvärinen and P. O. Hoyer. Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation*, 12(7):1705–1720, 2000.

[30] A. Hyvärinen and P. O. Hoyer. Emergence of topography and complex cell properties from natural images using extensions of ICA. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12, NIPS*99*, pages 827–833, Denver, Colorado, USA, Nov. 29 – Dec. 4, 1999, 2000. The MIT Press.

[31] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis.* John Wiley & Sons, 2001.

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

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

[34] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In Jordan [33], pages 105–161.

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

[36] J. Karhunen, S. Mălăroiu, and M. Ilmoniemi. Local linear independent component analysis using clustering. *International Journal of Neural Systems*, 10(6):439–451, 2000.

[37] R. E. Kass and L. Wasserman. Formal rules for selecting prior distributions: A review and annotated bibliography. Technical Report #583, Carnegie Mellon University, PA, 1994.

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

[39] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 3rd, extended edition, 2001.

[40] T. Kohonen, S. Kaski, and H. Lappalainen. Self-organized formation of various invariant-feature filters in the Adaptive-Subspace SOM. *Neural Computation*, 9(6):1321–1344, 1997.

[41] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 950–957, San Mateo, 1992. Morgan Kaufmann Publishers.

[42] J. Lampinen and A. Vehtari. Bayesian approach for neural networks - review and case studies. *Neural Networks*, 14(3):7–24, 2001.

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

[44] H. Lappalainen and J. W. Miskin. Ensemble learning. In M. Girolami, editor, *Advances in Independent Component Analysis*, pages 76–92. Springer-Verlag, Berlin, 2000.

[45] T. Lee and M. Lewicki. The generalized Gaussian mixture model using ICA. In *Proceedings of the 2nd International Workshop on Independent Component Analysis and Blind Source Separation (ICA2000)*, pages 239–244, Espoo, Finland, 2000.

[46] T. Lee, M. Lewicki, M. Girolami, and T. Sejnowski. Blind source separation of more sources than mixtures using overcomplete representations. *IEEE Signal Processing Letters*, 6:87–90, 1999.

[47] J. C. Lemm. Prior information and generalized questions. Technical Report A.I. Memo No. 1598, Massachusetts Institute of Technology, 1996.

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

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

[50] K. P. Murphy. A variational approximation for Bayesian networks with discrete and continuous latent variables. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pages 457–466, 1999.

[51] R. M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, 1992.

[52] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan [33], pages 355–368.

[53] E. Oja. PCA, ICA and nonlinear Hebbian learning. In *Proceedings of the ICANN'95*, pages 80–94, 1995.

[54] E. Oja and J. Karhunen. Signal separation by nonlinear Hebbian learning. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 83–87, 1995.

[55] L. Parra, C. Spence, and P. Sajda. Higher-order statistical properties arising from the non-stationarity of natural signals. In *Advances in Neural Information Processing Systems 13*, Denver, 2000.

[56] D.-T. Pham and J.-F. Cardoso. Blind separation of instantaneous mixtures of non stationary sources. In P. Pajunen and J. Karhunen, editors, *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, pages 187–192, Helsinki, Finland, June 19–22, 2000.

[57] T. Raiko and H. Valpola. Missing values in nonlinear factor analysis. In L. Zhang and F. Gu, editors, *Proceedings of the 8th International Conference on Neural Information Processing, ICONIP 2001*, volume 2, pages 822–827, Shanghai, China, 2001. Fudan University Press.

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

[59] M. Schervish. *Theory of Statistics.* Springer, New York, 1995.

[60] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

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

[62] H. Valpola. Nonlinear independent component analysis using ensemble learning: Theory. In P. Pajunen and J. Karhunen, editors, *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation, ICA 2000*, pages 251–256, Helsinki, Finland, June 19–22, 2000.

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

[64] H. Valpola, A. Honkela, and J. Karhunen. Nonlinear static and dynamic blind source separation using ensemble learning. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'01)*, Washington D.C., USA, 2001.

[65] H. Valpola and J. Karhunen. An unsupervised ensemble learning method for nonlinear dynamic state-space models. *Neural Computation*.

[66] H. Valpola, T. Raiko, and J. Karhunen. Building blocks for hierarchical latent variable models. In *Proceedings of the 3rd International Conference on Independent Component Analysis ans Blind Signal Separation, ICA 2001*, San Diego, California, USA, December 9–12, 2001. In press.

[67] M. Wainwright and E. Simoncelli. Scale mixtures of Gaussians and the statistics of natural images. In *Advances in Neural Information Processing Systems 12*, Cambridge, 2000.

[68] H. Wechsler, P. Phillips, V. Bruce, F. Soulie, and T. Huang. Face recognition: From theory to applications, 1998.

Appendix A

Cost Function of the Gaussian Variable

A node called the Gaussian variable was presented in Section 4.1. The part of the cost function C defined in (3.8) that corresponds to the variable is derived here. The posterior approximations of s, m and v are mutually independent

$$q(s, m, v) = q(s)q(m)q(v) \tag{A.1}$$

and they are of the form

$$q(s) = \mathrm{N}\left(s; \overline{s}, \widetilde{s}\right). \tag{A.2}$$

The first part C_p defined in (3.10) is addressed first.

$$\begin{aligned}
C_{s,p} &= -\left\langle \ln p\left(s \mid m, v\right)\right\rangle && \text{(A.3)}\\
&= -\left\langle \ln \mathrm{N}\left(s; m, \exp(-v)\right)\right\rangle && \text{(A.4)}\\
&= -\left\langle \ln \left[\left(2\pi \exp(-v)\right)^{-1/2} \exp \frac{-(s-m)^2}{2\exp(-v)}\right]\right\rangle && \text{(A.5)}\\
&= -\left\langle \ln \left[2\pi \exp(-v)\right]^{-1/2}\right\rangle - \left\langle \ln \exp\left[-\frac{1}{2}(s-m)^2 \exp v\right]\right\rangle && \text{(A.6)}\\
&= \frac{1}{2}\ln 2\pi - \frac{1}{2}\left\langle v\right\rangle + \frac{1}{2}\left\langle (s-m)^2\right\rangle \left\langle \exp v\right\rangle. && \text{(A.7)}
\end{aligned}$$

We still need to compute the expectation of $(s-m)^2$

$$\begin{aligned}
\left\langle (s-m)^2\right\rangle &= \left\langle s^2 - 2sm + m^2\right\rangle && \text{(A.8)}\\
&= \left\langle s^2\right\rangle - 2\left\langle sm\right\rangle + \left\langle m^2\right\rangle && \text{(A.9)}\\
&= \overline{s}^2 + \widetilde{s} - 2\overline{s}\left\langle m\right\rangle + \left\langle m\right\rangle^2 + \mathrm{Var}\left\{m\right\} && \text{(A.10)}\\
&= \left(\overline{s} - \left\langle m\right\rangle\right)^2 + \widetilde{s} + \mathrm{Var}\left\{m\right\}. && \text{(A.11)}
\end{aligned}$$

Substituting $\langle (s-m)^2 \rangle$ in (A.7) by (A.11) yields (4.1):

$$C_{s,p} = \frac{1}{2} \left\{ \langle \exp v \rangle \left[(\overline{s} - \langle m \rangle)^2 + \mathrm{Var}\left\{m\right\} + \widetilde{s} \right] - \langle v \rangle + \ln 2\pi \right\}. \tag{A.12}$$

The second part of the cost function C_q defined in (3.9) is

$$\begin{aligned}
C_{s,q} &= \langle \ln q(s) \rangle & \text{(A.13)} \\
&= \langle \ln \mathrm{N}\left(s; \overline{s}, \widetilde{s}\right) \rangle & \text{(A.14)} \\
&= \left\langle \ln \left[(2\pi\widetilde{s})^{-1/2} \exp \frac{-(s-\overline{s})^2}{2\widetilde{s}} \right] \right\rangle & \text{(A.15)} \\
&= -\frac{1}{2} \ln 2\pi\widetilde{s} + \frac{-\left\langle (s-\overline{s})^2 \right\rangle}{2\widetilde{s}} & \text{(A.16)} \\
&= -\frac{1}{2} \ln 2\pi\widetilde{s} + \frac{-\widetilde{s}}{2\widetilde{s}} & \text{(A.17)} \\
&= -\frac{1}{2} \ln 2\pi e\widetilde{s}, & \text{(A.18)}
\end{aligned}$$

which is the form used in Equation (4.2).

Appendix B

Update Rule of the Nonlinear Node

The update rule for the nonlinear node is considered in Section 4.4. It is a minimisation problem for the cost function $C_f = C_{f,p} + C_{f,q}$. Here is a proof that each iteration decreases the cost function (or keeps it the same if the partial derivatives vanish).

A different notation is used here. The quadratic term $a \langle f(s) \rangle + b[(\langle f(s) \rangle - f(s_{\text{current}}))^2 + \text{Var}\{f(s)\}] + d$ is taken into account by first finding an optimal expected value of f marked with f_{opt} and the variance σ_f^2 that corresponds to how much difference will cost.

$$\sigma_f^2 = (2b)^{-1} \tag{B.1}$$
$$f_{\text{opt}} = f(s_{\text{current}}) - \sigma_f^2 a \tag{B.2}$$

$$a \langle f(s) \rangle + b[(\langle f(s) \rangle - f(s_{\text{current}}))^2 + \text{Var}\{f(s)\}] + d \tag{B.3}$$
$$= \left\langle \frac{1}{2} \frac{(f(s) - f_{\text{opt}})^2}{\sigma_f^2} \right\rangle + d_2 \tag{B.4}$$

Now all the affected terms of C can be written as

$$C_{f,p} = \frac{1}{2} \left\{ \langle \exp v \rangle \left[(\bar{s} - \langle m \rangle)^2 + \text{Var}\{m\} + \tilde{s} \right] - \langle v \rangle \right\} + \frac{f_{\text{opt}}^2 - 2 f_{\text{opt}} \langle f(s) \rangle + \langle f(s)^2 \rangle}{2\sigma_f^2} \tag{B.5}$$

$$C_{f,q} = -\frac{1}{2} \ln(2e\pi\tilde{s}) \tag{B.6}$$

disregarding constants. The expectations $\langle f(s)\rangle$ and $\langle f(s)^2\rangle$ are

$$\langle f(s)\rangle = \exp\left(-\frac{\overline{s}^2}{2\widetilde{s}+1}\right)(2\widetilde{s}+1)^{-\frac{1}{2}} \tag{B.7}$$

$$\left\langle f(s)^2\right\rangle = \exp\left(-\frac{2\overline{s}^2}{4\widetilde{s}+1}\right)(4\widetilde{s}+1)^{-\frac{1}{2}} \tag{B.8}$$

and they are used as abbreviations.

Now we have to find $\overline{s}$ and $\widetilde{s}$ such that $C_f = C_{f,p} + C_{f,q}$ is minimized. The partial derivates of C_f are

$$\frac{\partial C_{f,p}}{\partial \overline{s}} = \langle \exp v\rangle (\overline{s} - \langle m\rangle) + \frac{\overline{s}}{\sigma_f^2}\left(\frac{2 f_{\text{opt}}\langle f(s)\rangle}{2\widetilde{s}+1} - \frac{2\left\langle f(s)^2\right\rangle}{4\widetilde{s}+1}\right) \tag{B.9}$$

$$\frac{\partial C_{f,q}}{\partial \overline{s}} = 0 \tag{B.10}$$

$$\frac{\partial C_{f,p}}{\partial \widetilde{s}} = \frac{\langle \exp v\rangle}{2} + \frac{(1 - 2\overline{s}^2 + 2\widetilde{s})\, f_{\text{opt}}\langle f(s)\rangle}{\sigma_f^2 (2\widetilde{s}+1)^2} - \frac{(1 - 4\overline{s}^2 + 4\widetilde{s})\left\langle f(s)^2\right\rangle}{\sigma_f^2 (4\widetilde{s}+1)^2} \tag{B.11}$$

$$\frac{\partial C_{f,q}}{\partial \widetilde{s}} = -\frac{1}{2\widetilde{s}}, \tag{B.12}$$

from which we get a fixed point iteration for the update candidate of $\widetilde{s}$

$$\frac{\partial C_f}{\partial \widetilde{s}} = -\frac{1}{2\widetilde{s}} + \frac{\partial C_{f,p}}{\partial \widetilde{s}} \tag{B.13}$$

$$\widetilde{s}_{new} = \left(2\frac{\partial C_{f,p}}{\partial \widetilde{s}}\right)^{-1}. \tag{B.14}$$

This would mean that the $\widetilde{s}$ is adjusted such that if $\partial C_{f,p}/\partial \widetilde{s}$ would stay the same, the whole partial derivative would become zero in one step. It is easy to see, that if $\partial C_f/\partial \widetilde{s}$ is positive, the iteration decreases $\widetilde{s}$ and vice versa. This means that the adjustments are always in the direction of the gradient descent.

It is worthwhile to note the connection derived from (4.1) and (4.2):

$$\frac{\partial^2 C_f}{\partial \langle m\rangle^2} = \langle \exp v\rangle = 2\frac{\partial C_f}{\partial \mathrm{Var}\{m\}}. \tag{B.15}$$

For $\overline{s}$, a gradient descent is used with the step length approximated from Newton's method. The approximation composed from (B.15)

$$\frac{\partial^2 C_f}{\partial \overline{s}^2} = \frac{\partial^2 C_{f,p}}{\partial \overline{s}^2} \approx 2\frac{\partial C_{f,p}}{\partial \widetilde{s}} = \frac{1}{\widetilde{s}_{new}} \tag{B.16}$$

$$\overline{s}_{new} = \overline{s} - \widetilde{s}_{new}\frac{\partial C_f}{\partial \overline{s}} \tag{B.17}$$

is accurate, if the the true posterior is a Gaussian. Since the step direction is derived directly from the derivative, it is always locally correct.

As was shown, these steps guarantee a direction, in which the cost function decreases locally. If the derivatives are zero, the iterating has converged. To guarantee also that the cost function does not increase because of a too long step, the update candidates are verified. The step is halved as long as the cost is about to rise.