In Advances in Independent Component Analysis,
ed. by Mark Girolami (Springer, 2000). In press.

Bayesian Nonlinear Independent Component Analysis by Multi-Layer Perceptrons

Harri Lappalainen and Antti Honkela

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

Summary. In this chapter, a nonlinear extension to independent component analysis is developed. The nonlinear mapping from source signals to observations is modelled by a multi-layer perceptron network and the distributions of source signals are modelled by mixture-of-Gaussians. The observations are assumed to be corrupted by Gaussian noise and therefore the method is more adequately described as nonlinear independent factor analysis. The nonlinear mapping, the source distributions and the noise level are estimated from the data. Bayesian approach to learning avoids problems with overlearning which would otherwise be severe in unsupervised learning with flexible nonlinear models.

1 Introduction

The linear principal and independent component analysis (PCA and ICA) model the data as having been generated by independent sources through a linear mapping. The difference between the two is that PCA restricts the distribution of the sources to be Gaussian, whereas ICA does not, in general, restrict the distribution of the sources.

In this chapter we introduce nonlinear counterparts of PCA and ICA where the generative mapping form sources to data is not restricted to be linear. The general form of the models discussed here is

$$\boldsymbol{x}(t) = f(\boldsymbol{s}(t)) + \boldsymbol{n}(t)\,. \tag{1}$$

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

Just as their linear counterparts, the nonlinear versions of PCA and ICA can be used for instance in dimension reduction and feature extraction. The difference between linear and nonlinear PCA is depicted in Fig. 1. In the linear PCA the data is described with a linear coordinate system whereas in the nonlinear PCA the coordinate system is nonlinear. The nonlinear PCA and ICA can be used for similar tasks as their linear counterparts, but they can be expected to capture the structure of the data better if the data points lie in a nonlinear manifold instead of a linear subspace.

Usually the linear PCA and ICA models do not have an explicit noise term $\boldsymbol{n}(t)$ and the model is thus simply

$$\boldsymbol{x}(t) = f(\boldsymbol{s}(t)) = \mathrm{A}\boldsymbol{s}(t) + \boldsymbol{b}\,. \tag{2}$$

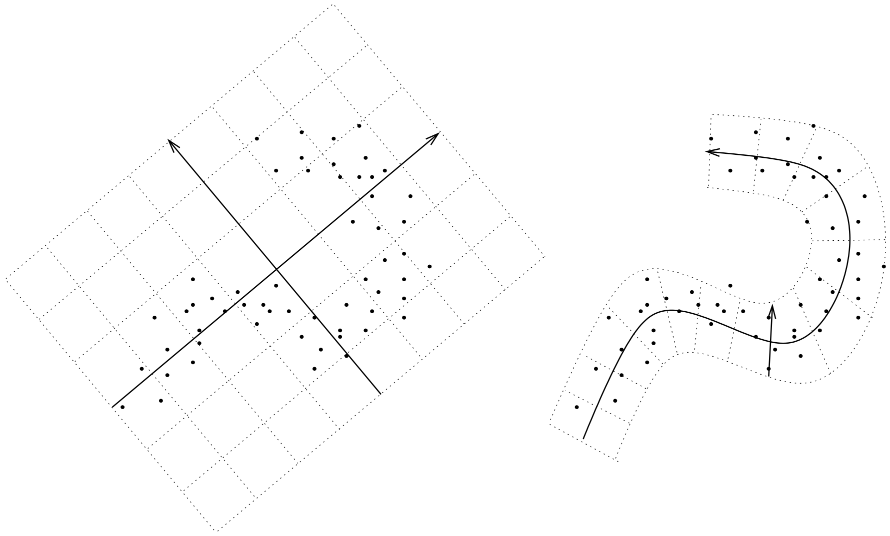

Fig. 1. On the left hand side the data is described with a linear coordinate system. On the right hand side the coordinate system is nonlinear

The corresponding PCA and ICA models which include the noise term are often called factor analysis and independent factor analysis (FA and IFA) models. The nonlinear models discussed here can therefore also be called nonlinear factor analysis and nonlinear independent factor analysis models.

In this chapter, the distribution of sources is modelled with Gaussian density in PCA and mixture-of-Gaussians density in ICA. Given enough Gaussians in the mixture, any density can be modelled with arbitrary accuracy using the mixture-of-Gaussians density, which means that the source density model is universal. Likewise, the nonlinear mapping $f()$ is modelled by a multi-layer perceptron (MLP) network which can approximate any nonlinear mapping with arbitrary accuracy given enough hidden neurons.

The noise on each observation channel (component of data vectors) is assumed to be independent and Gaussian, but the variance of the noise on different channels is not assumed to be equal. The noise could be modelled with a more general distribution, but we shall restrict the discussion to the simple Gaussian case. After all, noise is supposed to be something uninteresting and unstructured. If the noise is not Gaussian or independent, it is a sign of interesting structure which should be modelled by the generative mapping from the sources.

2 Choosing Among Competing Explanations

Each model with particular values for sources, parameters and noise terms can be considered as an explanation for the observations. Even with linear PCA and ICA there are infinitely many possible models which explain the observations completely. With flexible nonlinear models like an MLP network, the number of possible explanations is — loosely speaking — even higher (although mathematically speaking, ∞^2 would still be 'only' ∞).

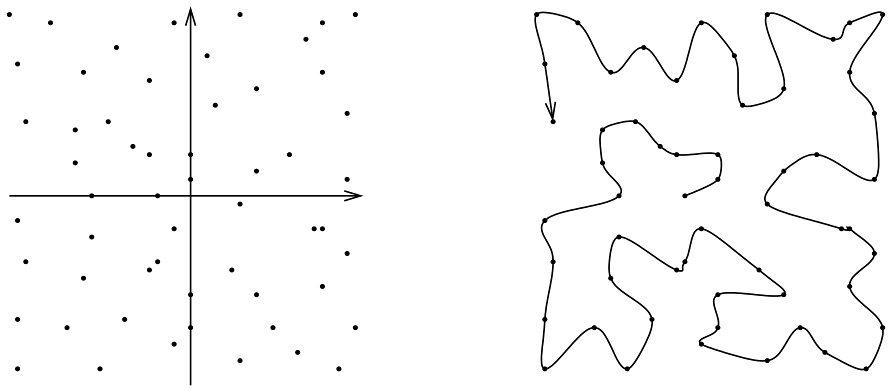

Fig. 2. The data is generated by two independent evenly distributed sources as shown on the left. Given enough hidden neurons, an MLP network is able to model the data as having been generated by a single source through a very nonlinear mapping, depicted on the right

An example of competing explanations is given in Fig. 2. The data is sampled from an even distribution inside a square. This is equivalent to saying that two independent sources, each evenly distributed, have generated the data as shown on the left hand side of the figure. If we only look at the probability of the data, the nonlinear mapping depicted on the right hand side of the figure is even better explanation as it gives very high probabilities to exactly those data points that actually occurred. However, it seems intuitively clear that the nonlinear model in Fig. 2 is much more complex than the available data would justify.

The exact Bayesian solution is that instead of choosing a single model, all models are used by weighting them according to their posterior probabilities. In other words, each model is taken into account in proportion with how probable they seem in light of the observations.

If we look at the predictions the above two models give about future data points, we notice that the more simple linear model with two sources predicts new points inside the square but the more complex nonlinear model with one source predicts new points only along the curved line. The prediction given by the more simple model is evidently closer to the prediction obtained by the exact Bayesian approach where the predictions of all models would be taken into account by weighting them according to the posterior probabilities of the models.

With complex nonlinear models like MLP networks, the exact Bayesian treatment is computationally intractable and we are going to resort to ensemble learning, which is discussed in Chap. 6. In ensemble learning, a computationally tractable parametric approximation is fitted to the posterior probabilities of the models.

In Sect. 6.4.2 it is shown that ensemble learning can be interpreted as finding the most simple explanation for the observations[1]. This agrees with the intuition that in Fig. 2, the simple linear model is better than the complex nonlinear model.

The fact that we are interested in simple explanations also explains why nonlinear ICA is needed at all if we can use nonlinear PCA. The nonlinearity of the mapping allows the PCA model to represent any time-independent probability density of the observations as originating from independent sources with Gaussian distributions. It would therefore seem that the non-Gaussian source models used in the nonlinear ICA cannot further increase the representational power of the model. However, for many naturally occurring processes the representation with Gaussian sources requires more complex nonlinear mappings than the representation with mixtures-of-Gaussians. Therefore the nonlinear ICA will often find a better explanation for the observations than the nonlinear PCA.

Similar considerations also explain why to use the MLP network for modelling the nonlinearity. Experience has shown that with MLP networks it is easy to model fairly accurately many naturally occurring multidimensional processes. In many cases the MLP networks give a more simple parametrisation for the nonlinearity than, for example, Taylor or Fourier series expansions.

On the other hand, it is equally clear that the ordinary MLP networks with sigmoidal nonlinearities are not the best models for all kinds of data. With the ordinary MLP networks it is, for instance, difficult to model mappings which have products of the sources. The purpose of this chapter is not to give the ultimate model for any data but rather to give a good model for many data, from which one can start building more sophisticated models by incorporating domain-specific knowledge. Most notably, the source models described here do not assume time-dependencies, which are often significant.

3 Nonlinear Factor Analysis

This section introduces a nonlinear counterpart of principal component analysis. As explained in Sect. 1, the model includes a noise term and we shall therefore call it nonlinear factor analysis. Learning is based on Bayesian ensemble learning which is introduced in Chap. 6. In order to keep the derivations simple, only Gaussian probability distributions are used which allows us to utilise many of the formulas derived in Sect. 6.6.1.

The posterior probability density of the unknown variables is approximated by a Gaussian distribution. As in Chap. 6, the variances of the Gaussian distributions of the model are parametrised by logarithm of standard deviation, log-std, because then the posterior distribution of these parameters will be closer to Gaussian which then agrees better with the assumption that the posterior is Gaussian.

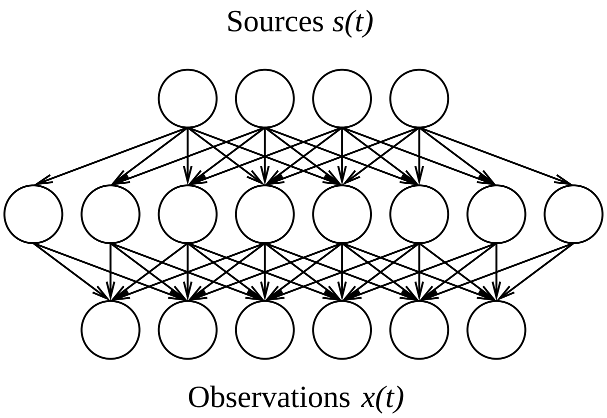

Fig. 3. The mapping from sources to observations is modelled by the familiar MLP network. The sources are on the top layer and observations in the bottom layer. The middle layer consists of hidden neurons each of which computes a nonlinear function of the inputs

3.1 Definition of the Model

The schematic structure of the mapping is shown in Fig. 3. The nonlinearity of each hidden neuron is the hyperbolic tangent, which is the same as the usual logistic sigmoid except for a scaling. The equation defining the mapping is

$$\boldsymbol{x}(t) = f(\boldsymbol{s}(t)) + \boldsymbol{n}(t) = \mathrm{B}\tanh(\mathrm{A}\boldsymbol{s}(t) + \boldsymbol{a}) + \boldsymbol{b} + \boldsymbol{n}(t)\,. \tag{3}$$

The matrices A and B are the weights of first and second layer and $\boldsymbol{a}$ and $\boldsymbol{b}$ are the corresponding biases.

The noise is assumed to be independent and Gaussian and therefore the probability distribution of $x(t)$ is

$$\boldsymbol{x}(t) \sim N(f(\boldsymbol{s}(t)), e^{2\boldsymbol{v}_x}) \tag{4}$$

Each component of the vector $\boldsymbol{v}_x$ gives the log-std of the corresponding component of $\boldsymbol{x}(t)$.

The sources are assumed to have zero mean Gaussian distributions and again the variances are parametrised by log-std $\boldsymbol{v}_s$.

$$\boldsymbol{s}(t) \sim N(0, e^{2\boldsymbol{v}_s}) \tag{5}$$

Since the variance of the sources can vary, variance of the weights A on the first layer can be fixed to a constant, which we choose to be one, without loosing any

[1] The complexity of the explanation is defined as the number of bits it takes to encode the observation using the model. In this case one would measure the total code length of the sources $\boldsymbol{s(t)}$, the parameters of the mapping and the noise $\boldsymbol{n(t)}$.

generality from the model. This is not case for the second layer weights. Due to the nonlinearity, the variances of the outputs of the hidden neurons are bounded from above and therefore the variance of the second layer weights cannot be fixed. In order to enable the network to shut off extra hidden neurons, the weights leaving one hidden neuron share the same variance parameter[2].

$$\mathrm{B} \sim N(0, e^{2\boldsymbol{v}_B}) \tag{6}$$

The elements of the matrix B are assumed to have a zero mean Gaussian distribution with individual variances for each column and thus the dimension of the vector $\boldsymbol{v}_B$ is the number of hidden neurons. Both biases $\boldsymbol{a}$ and $\boldsymbol{b}$ have Gaussian distributions parametrised by mean and log-std.

The distributions are summarised in (7)–(12).

$$\boldsymbol{x}(t) \sim N(f(\boldsymbol{s}(t)), e^{2\boldsymbol{v}_x}) \tag{7}$$
$$\boldsymbol{s}(t) \sim N(0, e^{2\boldsymbol{v}_s}) \tag{8}$$
$$\mathrm{A} \sim N(0, 1) \tag{9}$$
$$\mathrm{B} \sim N(0, e^{2\boldsymbol{v}_B}) \tag{10}$$
$$\boldsymbol{a} \sim N(m_a, e^{2v_a}) \tag{11}$$
$$\boldsymbol{b} \sim N(m_b, e^{2v_b}) \tag{12}$$

The distributions of each set of log-std parameters are modelled by Gaussian distributions whose parameters are usually called hyperparameters.

$$\boldsymbol{v}_x \sim N(m_{v_x}, e^{2v_{v_x}}) \tag{13}$$
$$\boldsymbol{v}_s \sim N(m_{v_s}, e^{2v_{v_s}}) \tag{14}$$
$$\boldsymbol{v}_B \sim N(m_{v_B}, e^{2v_{v_B}}) \tag{15}$$

The prior distributions of m_a, v_a, m_b, v_b and the six hyperparameters $m_{v_s}, \ldots, v_{v_B}$ are assumed to be Gaussian with zero mean and standard deviation 100, i.e., the priors are assumed to be very flat.

3.2 Cost Function

In ensemble learning, the goal is to approximate the posterior pdf of all the unknown values in the model. Let us denote the observations by X. Everything else in the model is unknown, i.e., the sources, parameters and hyperparameters. Let us denote all these unknowns by the vector $\boldsymbol{\theta}$. The cost function measures the misfit between the actual posterior pdf $p(\boldsymbol{\theta}|\mathrm{X})$ and its approximation $q(\boldsymbol{\theta}|\mathrm{X})$.

[2] A hidden neuron will be shut off if all leaving weights are close to zero. Thinking in coding terms, it is easier for the network to encode this in one variance parameter than to encode it independently for all the weights.

The posterior is approximated as a product of independent Gaussian distributions

$$q(\boldsymbol{\theta}|\mathrm{X}) = \prod_i q(\theta_i|\mathrm{X}) \,. \tag{16}$$

Each individual Gaussian $q(\theta_i|\mathrm{X})$ is parametrised by the posterior mean $\bar{\theta}_i$ and variance $\tilde{\theta}_i$ of the parameter.

The functional form of the cost function $C_{\boldsymbol{\theta}}(\mathrm{X}; \bar{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}})$ is given in Chap. 6. The cost function can be interpreted to measure the misfit between the actual posterior $p(\boldsymbol{\theta}|\mathrm{X})$ and its factorial approximation $q(\boldsymbol{\theta}|\mathrm{X})$. It can also be interpreted as measuring the number of bits it would take to encode X when approximating the posterior pdf of the unknown variables by $q(\boldsymbol{\theta}|\mathrm{X})$.

The cost function is minimised with respect to the posterior means $\bar{\theta}_i$ and variances $\tilde{\theta}_i$ of the unknown variables θ_i. The end result of the learning is therefore not just an estimate of the unknown variables, but a distribution over the variables.

The simple factorising form of the approximation $q(\boldsymbol{\theta}|\mathrm{X})$ makes the cost function computationally tractable. The cost function can be split into two terms, C_q and C_p, where the former is an expectation over $\ln q(\boldsymbol{\theta}|\mathrm{X})$ and the latter is an expectation over $-\ln p(\mathrm{X}, \boldsymbol{\theta})$.

It turns out that the term C_q is not a function of the posterior means $\bar{\theta}_i$ of the parameters, only the posterior variances. It has a similar term for each unknown variable.

$$C_q(\mathrm{X}; \tilde{\boldsymbol{\theta}}) = \sum_i -\frac{1}{2} \ln 2\pi e \tilde{\theta}_i \tag{17}$$

Most of the terms of C_p are also trivial. The Gaussian densities in (8)–(15) yield terms of the form

$$-\ln p(\theta) = \frac{1}{2}(\theta - m_\theta)^2 e^{-2v_\theta} + v_\theta + \frac{1}{2}\ln 2\pi \tag{18}$$

Since θ, m_θ and v_θ are independent in q, the expectation over q yields

$$\frac{1}{2}[(\bar{\theta} - \bar{m}_\theta)^2 + \tilde{\theta} + \tilde{m}_\theta] e^{2\tilde{v}_\theta - 2\bar{v}_\theta} + \bar{v}_\theta + \frac{1}{2}\ln 2\pi \,. \tag{19}$$

Only the term originating from (7) needs some elaboration. Equation (7) yields

$$-\ln p(x) = \frac{1}{2}(x - f)^2 e^{-2v_x} + v_x + \frac{1}{2}\ln 2\pi \tag{20}$$

and the expectation over q is

$$\frac{1}{2}[(x - \bar{f})^2 + \tilde{f}] e^{2\tilde{v}_x - 2\bar{v}_x} + \bar{v}_x + \frac{1}{2}\ln 2\pi \tag{21}$$

The rest of this section is dedicated on evaluating the posterior mean $\bar{f}$ and variance $\tilde{f}$ of the function f. We shall begin from the sources and weights and show how the posterior mean and variance can be propagated through the network yielding the needed posterior mean and variance of the function f at the output. The effect of nonlinearities g of the hidden neurons are approximated by first and second order Taylor's series expansions around the posterior mean. Apart from that, the computation is analytical.

The function f consists of two multiplications with matrices and a nonlinearity in between. The posterior mean and variance for a product $u = yz$ are

$$\bar{u} = E\{u\} = E\{yz\} = E\{y\}E\{z\} = \bar{y}\bar{z} \tag{22}$$

and

$$\tilde{u} = E\{u^2\} - \bar{u}^2 = E\{y^2z^2\} - (\bar{y}\bar{z})^2 = \\ E\{y^2\}E\{z^2\} - \bar{y}^2\bar{z}^2 = (\bar{y}^2 + \tilde{y})(\bar{z}^2 + \tilde{z}) - \bar{y}^2\bar{z}^2 \,, \tag{23}$$

given that y and z are posteriorly independent. According to the assumption of the factorising form of $q(\boldsymbol{\theta}|\mathrm{X})$, the sources and the weights are independent and we can use the above formulas. The inputs going to hidden neurons consist of sums of products of weights and sources, each posteriorly independent, and it is therefore easy compute the posterior mean and variance of the inputs going to the hidden neurons; both the means and variances of a sum of independent variables add up.

Let us now pick one hidden neuron having nonlinearity g and input ξ, i.e., the hidden neuron is computing $g(\xi)$. At this point we are not assuming any particular form of g although we are going to use $g(\xi) = \tanh\xi$ in all the experiments; the following derivation is general and can be applied to any sufficiently smooth function g.

In order to be able to compute the posterior mean and variance of the function g, we are going to apply the Taylor's series expansion around the posterior mean $\bar{\xi}$ of the input. We choose the second order expansion when computing the mean and the first order expansion when computing the variance. The choice is purely practical; higher order expansions could be used as well but these are the ones that can be computed from the posterior mean and variance of the inputs alone.

$$\bar{g}(\xi) \approx g(\bar{\xi}) + \frac{1}{2}g''(\bar{\xi})\tilde{\xi} \tag{24}$$

$$\tilde{g}(\xi) \approx [g'(\bar{\xi})]^2\tilde{\xi} \tag{25}$$

After having evaluated the outputs of the nonlinear hidden neurons, it would seem that most of the work has already been done. After all, it was already shown how to compute the posterior mean and variance of a weighted sum and the outputs of the network will be weighted sums of the outputs of the hidden neurons. Unfortunately, this time the terms in the sum are no longer independent. The sources are posteriorly independent by virtue of the approximation $q(\boldsymbol{\theta}|\mathrm{X})$, but the values of the hidden neurons are posteriorly dependent which enforces us to use a more complicated scheme for computing the posterior variances of these weighted sums. The posterior means will be as simple as before, though.

The reason for the outputs of the hidden neurons to be posteriorly dependent is that the value of one source can potentially affect all the outputs. This is illustrated in Fig. 4. Each source affects the output of the whole network through several paths and in order to be able to determine the variance of the outputs, the paths originating from different sources need to be kept separate. This is done by

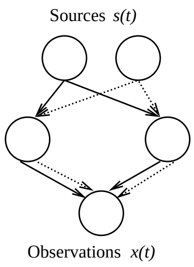

Fig. 4. The converging paths from two sources are shown. Both input neurons affect the output neuron through two paths going through the hidden neurons. This means that the posterior variances of the two hidden neurons are neither completely correlated nor uncorrelated and it is impossible to compute the posterior variance of the output neuron without keeping the two paths separate. Effectively this means computing the Jacobian matrix of the output with respect to the inputs

keeping track of the partial derivatives $\frac{\partial g(\xi)}{\partial s_i}$. Equation (26) shows how the total posterior variance of the output $g(\xi)$ of one of the hidden neurons can be split into terms originating from each source plus a term $\tilde{g}^*(\xi)$ which contains the variance originating from the weights and biases, i.e., those variables which affect any one output through only a single path.

$$\tilde{g}(\xi) = \tilde{g}^*(\xi) + \sum_i \tilde{s}_i \left[\frac{\partial g(\xi)}{\partial s_i}\right]^2 \tag{26}$$

When the outputs are multiplied by weights, it is possible to keep track of how this affects the posterior mean, the derivatives w.r.t. the sources and the variance originating from other variables than the sources, i.e., from weights and biases. The total variance of the output of the network is then obtained by

$$\tilde{f} = \tilde{f}^* + \sum_i \tilde{s}_i \left[\frac{\partial f}{\partial s_i}\right]^2 , \tag{27}$$

where f denotes the components of the output and we have computed the posterior variance of the outputs of the network which is needed in (21). To recapitulate what is done, the contributions of different sources to the variances of the outputs of the network are monitored by computing the Jacobian matrix of the outputs w.r.t. the sources and keeping this part separate from the variance originating from other variables.

The only approximations done in the computation are the ones approximating the effect of nonlinearity. If the hidden neurons were linear, the computation would be exact. The nonlinearity of the hidden neurons is delt with by linearising around the posterior mean of the inputs of the hidden neurons. The smaller the variances the more accurate this approximation is. With increasing nonlinearity and variance of the inputs, the approximation gets worse.

Compared to ordinary forward phase of an MLP network, the computational complexity is greater by about a factor of $5N$, where N is the number of sources. The factor five is due to propagating distributions instead of plain values. The need to keep the paths originating from different sources separate explains the factor N. Fortunately, much of the extra computation can be made into good use later on when adapting the distributions of variables.

3.3 Update Rules

Any standard optimisation algorithm could be used for minimising the cost function $C(\mathrm{X}; \bar{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}})$ with respect to the posterior means $\bar{\boldsymbol{\theta}}$ and variances $\tilde{\boldsymbol{\theta}}$ of the unknown variables. As usual, however, it makes sense utilising the particular structure of the function to be minimised.

Those parameters which are means or log-std of Gaussian distributions, e.g., m_b, m_{v_B}, v_a and v_{v_x}, can be solved in the same way as the parameters of Gaussian distribution where solved in Sect. 6.1. Since the parameters have Gaussian priors, the equations do not have analytical solutions, but Newton-iteration can be used. For each Gaussian, the posterior mean and variance of the parameter governing the mean is solved first by assuming all other variables constant and then the same thing is done for the log-std parameter, again assuming all other variables constant.

Since the mean and variance of the output of the network and thus also the cost function was computed layer by layer, it is possible to use the ordinary back-propagation algorithm to evaluate the partial derivatives of the part C_p of the cost function w.r.t. the posterior means and variances of the sources, weights and biases. Assuming the derivatives computed, let us first take a look at the posterior variances $\tilde{\theta}$.

The effect of the posterior variances $\tilde{\theta}$ of sources, weights and biases on the part C_p of the cost function is mostly due to the effect on $\tilde{f}$ which is usually very close to linear (this was also the approximation made in the evaluation of the cost function). The terms $\tilde{f}$ have a linear effect on the cost function, as is seen in (21), which means that the over all effect of the terms $\tilde{\theta}$ on C_p is close to linear. The partial derivative of C_p with respect to $\tilde{\theta}$ is therefore roughly constant and it is reasonable to use the following fixed point equation to update the variances:

$$0 = \frac{\partial C}{\partial \tilde{\theta}} = \frac{\partial C_p}{\partial \tilde{\theta}} + \frac{\partial C_q}{\partial \tilde{\theta}} = \frac{\partial C_p}{\partial \tilde{\theta}} - \frac{1}{2\tilde{\theta}} \Rightarrow \tilde{\theta} = \frac{1}{2\frac{\partial C_p}{\partial \tilde{\theta}}} . \tag{28}$$

The remaining parameters to be updated are the posterior means $\bar{\theta}$ of the sources, weights and biases. For those parameters it is possible to use Newton iteration since the corresponding posterior variances $\tilde{\theta}$ actually contain the information about the second order derivatives of the cost function C w.r.t. $\bar{\theta}$. It holds

$$\tilde{\theta} \approx \frac{1}{\frac{\partial^2 C}{\partial \bar{\theta}^2}} \tag{29}$$

and thus the step in Newton iteration can be approximated

$$\bar{\theta} \leftarrow \bar{\theta} - \frac{\frac{\partial C_p}{\partial \bar{\theta}}}{\frac{\partial^2 C}{\partial \bar{\theta}^2}} \approx \bar{\theta} - \frac{\partial C_p}{\partial \bar{\theta}} \tilde{\theta} . \tag{30}$$

Equation (29) would be exact if the posterior pdf $p(\boldsymbol{\theta}|\mathrm{X})$ were exactly Gaussian. This would be true if the mapping f were linear. The approximation in (29) is therefore good as long as the function f is roughly linear around the current estimate of $\bar{\boldsymbol{\theta}}$.

Avoiding Problems Originating from Approximation of the Nonlinearity of the Hidden Neurons. The approximations in (24) and (25) can give rise to problems with ill defined posterior variances of sources or first layer weights A or biases $\boldsymbol{a}$. This is because the approximations take into account only local behaviour of the nonlinearities g of the hidden neurons. With MLP networks the posterior is typically multimodal and therefore, in a valley between two maxima, it is possible that the second order derivative of the logarithm of the posterior w.r.t. a parameter θ is positive. This means that the derivative of the C_p part of the cost function with respect to the posterior variance $\tilde{\theta}$ of that parameter is negative, leading to a negative estimate of variance in (28).

It is easy to see that the problem is due to the local estimate of g since the logarithm of the posterior eventually has to go to negative infinity. The derivative of the C_p term w.r.t. the posterior variance $\tilde{\theta}$ will thus be positive for large $\tilde{\theta}$, but the local estimate of g fails to account for this.

In order to discourage the network from adapting itself to areas of parameter space where the problems might occur and to deal with the problem if it nevertheless occurred, the terms in (24) which give rise to negative derivative of C_p with respect to $\tilde{\theta}$ will be neglected in the computation of the gradients. As this can only make the estimate of $\tilde{\theta}$ in (28) smaller, this leads, in general, to increasing the accuracy of the approximations in (24) and (25).

Stabilising the Fixed-Point Update Rules. The adaptations rules in (28) and (30) assume other parameters to be constant. The weights, sources and biases are updated all at once, however, because it would not be computationally efficient to update only one at a time. The assumption of independence is not necessarily valid, particularly for the posterior means of the variables, which may give rise to instabilities. Several variables can have a similar effect on outputs and when they are all updated to the values which would be optimal given that the others stay constant, the combined effect is too large.

This type of instability can be detected by monitoring the directions of updates of individual parameters. When the problem of correlated effects occurs, consecutive updated values start oscillating. A standard way to dampen these oscillations in fixed point algorithms is to introduce a learning parameter α for each parameter and update it according to the following rule:

$$\alpha \leftarrow \begin{cases} 0.8\alpha & \text{if sign of change was same} \\ \min(1, 1.05\alpha) & \text{if sign of change was different} \end{cases} \tag{31}$$

This gives the following fixed point update rules for the posterior means and variances of the sources, weights and the biases:

$$\bar{\theta} \leftarrow \bar{\theta} - \alpha_{\bar{\theta}} \frac{\partial C_p}{\partial \bar{\theta}} \tilde{\theta} \tag{32}$$

$$\tilde{\theta} \leftarrow \frac{1}{\left[2 \frac{\partial C_p}{\partial \tilde{\theta}}\right]^{\alpha_{\tilde{\theta}}}} \tilde{\theta}^{1-\alpha_{\tilde{\theta}}} \tag{33}$$

The reason why a weighted geometric rather than arithmetic mean is applied to the posterior variances is that variance is a scale parameter. The relative effect of adding a constant to the variance varies with the magnitude of the variance whereas the relative effect of multiplying by a constant is invariant.

Using Additional Information for Updating Sources. With sources, it is possible to measure and compensate some of the correlated effects in the updates. Recall that the Jacobian matrix of the output $\boldsymbol{f}$ of the network w.r.t. the sources was computed when taking into account the effects of multiple paths of propagating the values of sources. This will be used to compensate the assumption of independent updates, in addition to the learning rates α.

Suppose we have two sources whose effect on outputs are positively correlated. Assuming the effects independent means that the step will be to large and the actual step size should be less than what the Newton iteration suggests. This can be detected from computing the change resulting in the outputs and projecting it back for each source independently to see how much each source alone should change to produce the same change in the outputs. The difference between the change of one source in the update and change resulting from all the updates can then be used to adjust the step sizes in the update.

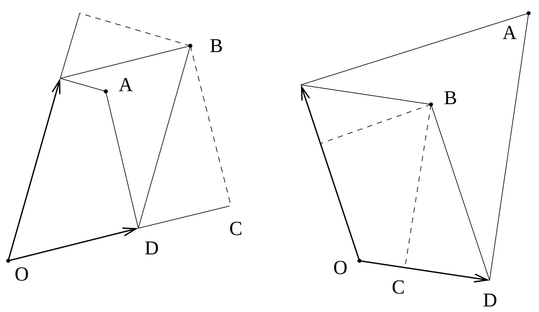

Fig. 5. Illustration of the correction of error resulting from assuming independent updates of the sources. The figures show the effect two sources have on the outputs. On the left hand side the effects of sources on $\boldsymbol{x}$ are positively correlated and consequently the step sizes are overestimated. On the right hand side the effects are negatively correlated and the step sizes are underestimated

Two examples of correction are depicted in Fig. 5. The left hand side graph shows a case where the effects of sources on the outputs are positively correlated and the right hand side graph has negatively correlated effects. Current output of the network is in the origin O and the minimum of the cost function is in point A. Black arrows show where the output would move if the sources were minimised independently. The combined updates would then take the output to point B.

As the effects of sources on $\boldsymbol{x}$ are correlated, point B, the resulting overall change in $\boldsymbol{x}$, differs from point A. Projecting the point B back to the sources, comparison between the resulting step size C and the desired step size D can be used for adjusting the step size. The new step size on the source would be D/C times the original. With positively correlated effects the adjusting factor D/C is less then one, but with negatively correlated sources it is greater than one. For the sake of stability, the corrected step is restricted to be at most twice the original.

4 Nonlinear Independent Factor Analysis

The nonlinear factor analysis model introduced in the previous section has Gaussian distributions for the sources. In this section we are going to show how that model can easily be extended to have mixture-of-Gaussians models for sources. In doing so we are largely following the method introduced in [1] for Bayesian linear independent factor analysis. The resulting model is a nonlinear counterpart of ICA or, more accurately, a nonlinear counterpart of independent factor analysis because the model includes finite noise. The difference between the models is similar to that between linear PCA and ICA because the first layer weight matrix A in the network has the same indeterminacies in nonlinear PCA as in linear PCA. The indeterminacy is discussed in the introductory chapter.

According to the model for the distribution of the sources, there are several Gaussian distributions and at each time instant, the source originates from one of them. Let us denote the index of the Gaussian from which the source $s_i(t)$ originates by $M_i(t)$. The model for the distribution for the ith source at time t is

$$p(s_i(t)|\theta) = \sum_{M_i(t)} P(M_i(t)|\theta) p(s_i(t)|\theta, M_i(t)), \tag{34}$$

where $p(s_i(t)|\theta, M_i(t) = j)$ is a time-independent Gaussian distribution with its own mean m_{ij} and log-std v_{ij}. The probabilities $P(M_i(t)|\theta)$ of different Gaussians are modelled with time-independent soft-max distributions.

$$P(M_i(t) = j|\theta) = \frac{e^{c_{ij}}}{\sum_{j'} e^{c_{ij'}}} \tag{35}$$

Each combination of different Gaussians producing the sources can be considered a different model. The number of these models is enormous, of course, but their posterior distribution can still be approximated by a similar factorial approximation which is used for other variables.

$$Q(\mathrm{M}|\mathrm{X}) = \prod_{M_i(t)} Q(M_i(t)|\mathrm{X}) \tag{36}$$

Without losing any further generality, we can now write

$$q(s_i(t), M_i(t)|\theta) = Q(M_i(t)|\theta) q(s_i(t)|\theta, M_i(t)), \tag{37}$$

which yields

$$q(s_i(t)|\theta) = \sum_j q(M_i(t) = j|\theta) Q(s_i(t)|\theta, M_i(t) = j). \tag{38}$$

This means that the approximating ensemble for the sources has a form similar to the prior, i.e., an independent mixture of Gaussians, although the posterior mixture is different at different times.

Due to the assumption of factorial posterior distribution of the models, the cost function can be computed as easily as before. Let us denote $Q(M_i(t) = j|\theta) = \dot{s}_{ij}(t)$

and the posterior mean and variance of $q(s_i(t)|\theta, M_i(t) = j)$ by $\bar{s}_{ij}(t)$ and $\tilde{s}_{ij}(t)$. It easy to see that the posterior mean and variance of $s_i(t)$ are

$$\bar{s}_i(t) = \sum_j \dot{s}_{ij}(t)\bar{s}_{ij}(t) \tag{39}$$

$$\tilde{s}_i(t) = \sum_j \dot{s}_{ij}(t)[\tilde{s}_{ij}(t) + (\bar{s}_{ij}(t) - \bar{s}_i(t))^2] \,. \tag{40}$$

After having computed the posterior mean $\bar{s}_i$ and variance $\tilde{s}_i$ of the sources, the computation of the C_p part of the cost function proceeds as with nonlinear factor analysis in the previous section. The C_q part yields terms of the form

$$\begin{aligned}
&q(s_i(t)|\mathrm{X}) \ln q(s_i(t)|\mathrm{X}) = \\
&\sum_j \dot{s}_{ij}(t) q(s_i(t)|M_i(t) = j, \mathrm{X}) \ln \sum_j \dot{s}_{ij}(t) q(s_i(t)|M_i(t), \mathrm{X}) = \\
&\sum_j \dot{s}_{ij}(t) q(s_i(t)|M_i(t) = j, \mathrm{X}) \ln \dot{s}_{ij}(t) q(s_i(t)|M_i(t), \mathrm{X}) = \\
&\qquad \sum_j \dot{s}_{ij}(t)[\ln \dot{s}_{ij}(t) + q(s_i(t)|M_i(t) = j, \mathrm{X}) \ln q(s_i(t)|M_i(t) = j, \mathrm{X})]
\end{aligned} \tag{41}$$

and we have thus reduced the problem to a previously solved one. The terms $q(s_i(t)|M_i(t), \mathrm{X}) \ln q(s_i(t)|M_i(t), \mathrm{X})$ are the same as for the nonlinear factor analysis and otherwise the equation has the same form as in model selection in Chap. 6. This is not surprising since the terms $Q(M_i(t)|\mathrm{X})$ are the probabilities of different models and we are, in effect, therefore doing factorial model selection.

Most update rules are the same as for nonlinear factor analysis. Equations (39) and (40) bring the terms $\dot{s}_{ij}(t)$ for updating the means m_{ij} and log-std parameters v_{ij} of the sources. It turns out that they both will be weighted with $\dot{s}_{ij}$, i.e., the observation is used for adapting the parameters in proportion to the posterior probability of that observation originating from that particular Gaussian distribution.

5 Experiments

5.1 Learning Scheme

The learning scheme for all the experiments was the same. First, linear PCA is used to find sensible initial values for the posterior means of the sources. The method was chosen because it has given good results in initialising the model vectors of a self-organising map (SOM). The posterior variances of the sources are initialised to small values. Good initial values are important for the method since the network can effectively prune away unused parts as will be seen in the experiments later on. Initially the weights of the network have random values and the network has quite bad representation for the data. If the sources were adapted from random values also, the network would consider many of the sources useless for the representation and prune them away. This would lead to a local minimum from which the network would not recover.

Therefore the sources are fixed at the values given by linear PCA for the first 50 iterations through the whole data. This is long enough for the network to find a meaningful mapping from sources to the observations, thereby justifying using the sources for the representation. For the same reason, the parameters controlling the distributions of the sources, weights, noise and the hyperparameters are not adapted during the first 100 iterations. They are adapted only after the network has found sensible values for the variables whose distributions these parameters control.

In all simulations, the total number of iterations is 7500, where one iteration means going through all the observations. For nonlinear independent factor analysis simulations, a nonlinear subspace is estimated with 2000 iterations by the nonlinear factor analysis after which the sources are rotated with a linear ICA algorithm. In these experiments, FastICA was used [4]. The rotation of the sources is compensated by an inverse rotation to the first layer weight matrix A. The nonlinear independent factor analysis algorithm is then applied for the remaining 5500 iterations.

5.2 Helix

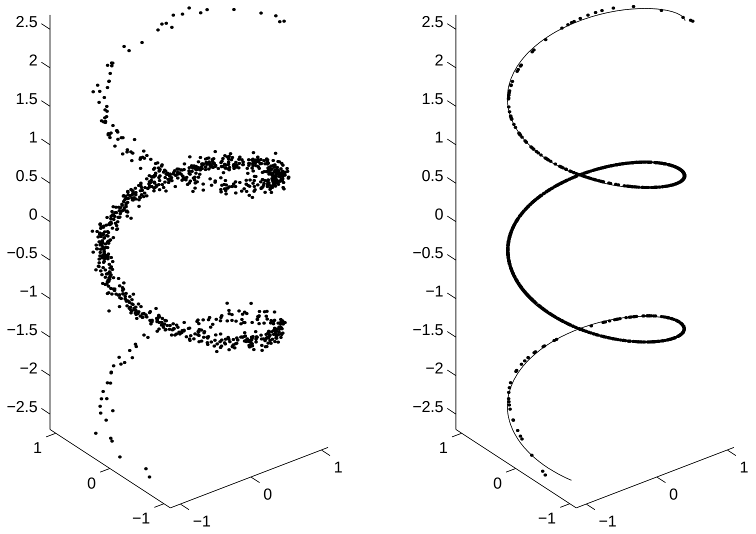

Fig. 6. The plot on the left shows the data points and the plot on the right shows the reconstructions made by the network together with the underlying helical subspace. The MLP network has clearly been able to find the underlying one-dimensional nonlinear subspace where the data points lie

Let us first take a look at a toy problem which shows that it is possible to find a nonlinear subspace and model it with an MLP network in an unsupervised manner. A set of 1000 data points, shown on the left plot of Fig. 6, were generated from a normally distributed source s into a helical subspace. The z-axis had a linear correspondence to the source and the x- and y-axes were sine and cosine: $x = \sin(\pi s)$, $y = \cos(\pi s)$ and $z = s$. Gaussian noise with standard deviation 0.05 was added to all three data components.

One-dimensional nonlinear subspaces were estimated with the nonlinear independent factor analysis algorithm. Several different numbers of hidden neurons and initialisations of the MLP networks were tested and the network which minimised the cost function was chosen. The best network had 16 hidden neurons. The original noisy data and the means of the outputs of the best MLP network are shown in Fig. 6. It is evident that the network was able to learn the correct subspace. Only the tails of the helix are somewhat distorted. The network estimated the standard deviations of the noise on different data components to be 0.052, 0.055 and 0.050. This is in close correspondence with the actual noise level of 0.05.

This problem is not enough to demonstrate the advantages of the method since it does not prove that the method is able to deal with high dimensional latent spaces. This problem was chosen simply because it is easy to visualise.

5.3 Nonlinear Artificial Data

Gaussian Sources. The following experiments with nonlinear factor analysis algorithm demonstrate the ability of the network to prune away unused parts. The data was generated from five normally distributed sources through a nonlinear mapping. The mapping was generated by a randomly initialised MLP network having 20 hidden neurons and ten output neurons. Gaussian noise with standard deviation of 0.1 was added to the data. The nonlinearity for the hidden neurons was chosen to be the inverse hyperbolic sine, which means that the nonlinear factor analysis algorithm using MLP network with tanh-nonlinearities cannot use exactly the same weights.

Figure 7 shows how much of the energy remains in the data when a number of linear PCA components are extracted. This measure is often used to deduce the linear dimension of the data. As the figure shows, there is no obvious turn in the curve and it would be impossible to deduce the nonlinear dimension.

With the nonlinear factor analysis by MLP networks, not only the number of the sources but also the number of hidden neurons in the MLP network needs to be estimated. With the Bayesian approach this is not a problem, as is shown in Figs. 8 and 9. The cost function exhibits a broad minimum as a function of hidden neurons and a saturating minimum as a function of sources. The reason why the cost function saturates as a function of sources is that the network is able to effectively prune away unused sources. In the case of ten sources, for instance, the network actually uses only five of them.

The pressure to prune away hidden neurons is not as big which can be seen in Fig. 10. A reliable sign of pruning is the amount of bits which the network uses for describing a variable. Recall that it was shown in Sect. 6.4.2 that the cost function can be interpreted as the description length of the data. The description length can also be computed for each variable separately and this is shown in Fig. 10. The

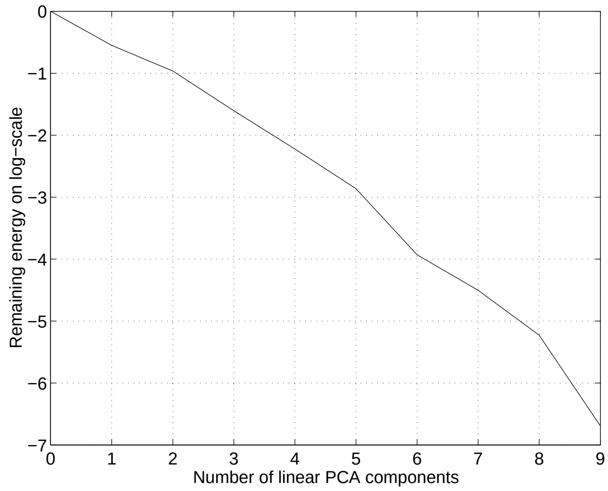

Fig. 7. The graph shows the remaining energy in the data as a function of the number of extracted linear PCA components. The total energy is normalised to unity (zero on logarithmic scale). The data has been generated from five Gaussian sources but as the mapping is nonlinear, the linear PCA cannot be used for finding the original subspace

MLP network had seven input neurons, i.e., seven sources, and 100 hidden neurons. The upper left plot shown clearly that the network effectively uses only five of the sources and very few bits are used to describe the other two sources. This is evident also from the first layer weight matrix A on the upper right plot, which shows the average description length of the weights leaving each input neuron.

The lower plot of Fig. 10 also shows the average description length of the weight matrix A, but now the average is taken row-wise and thus tells how many bits are used for describing the weights arriving to each hidden neuron. It appears that about six or seven hidden neurons have been pruned away, but the pruning is not as complete as in the case of sources. This is because for each source the network has to represent 1000 values, one for each observation vector, but for each hidden neuron the network only needs to represent five plus twenty (the effective number of inputs and outputs) values and there is thus much less pressure to prune away a hidden neuron.

Non-Gaussian Sources. The following experiments demonstrate the ability of the nonlinear independent factor analysis algorithm to find the underlying latent variables which have generated the observations.

In these experiments, a similar scheme was used to generate data as with the Gaussian sources before. Now a total of eight sources was used with four sub-Gaussian and four super-Gaussian sources. The generating MLP network was also larger, having 30 hidden neurons and 20 output neurons. The super-Gaussian

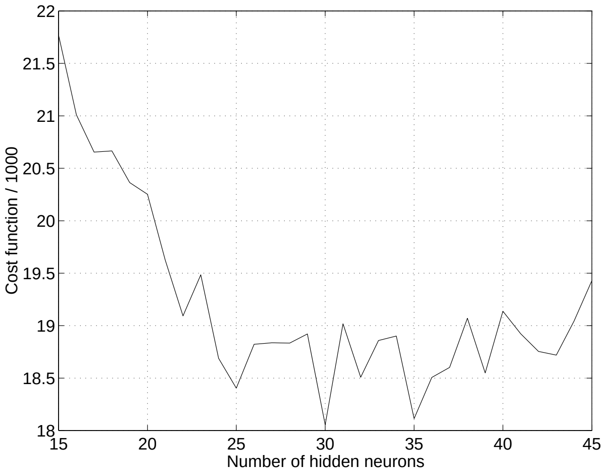

Fig. 8. The value of the cost function is shown as a function of the number of hidden neurons in the MLP network modelling the nonlinear mapping from five sources to the observations. Ten different initialisations were tested to find the minimum value for each number of hidden neurons. The cost function exhibits a broad and somewhat noisy minimum. The smallest value for the cost function was obtained using 30 hidden neurons

sources were generated by taking a hyperbolic sine, $\sinh x$, from a normally distributed random variable and then normalising the variance to unity. In generating sub-Gaussian sources, inverse hyperbolic sine, $\sinh^{-1} x$, was used instead of $\sinh x$.

Several different numbers of hidden neurons were tested in order to optimise the structure of the network, but the number of sources was assumed to be known. This assumption is reasonable since it is possible to optimise the number of sources simply by minimising the cost function as the experiments with the Gaussian sources show. The network which minimised the cost function turned out to have 50 hidden neurons. The number of Gaussians in each of the mixtures modelling the distribution of each source was chosen to be three and no attempt was made to optimise this.

The results are depicted in Figs. 11, 12 and 13. Each figure shows eight scatter plots, each of which corresponds to one of the eight sources. The original source which was used for generating the data is on the x-axis and the estimated source in on the y-axis of each plot. Each point corresponds to one data vector. The upper plots of each figure correspond to the super-Gaussian and the lower plots to the sub-Gaussian sources. Optimal result would be a straight line which would mean that the estimated values of the sources coincide with the true values.

Figure 11 shows the result of a linear FastICA algorithm [4]. The linear ICA is able to retrieve the original sources with 0.7 dB signal to noise ratio. In practise a linear method could not deduce the number of sources and the result would be even

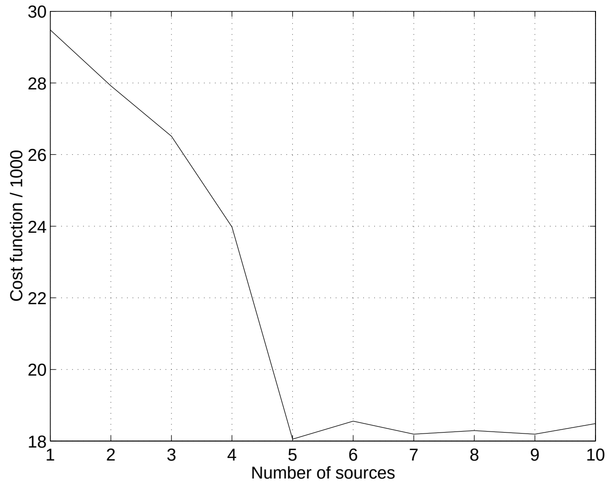

Fig. 9. The value of the cost function is shown as a function of the number of sources. The MLP network had 30 hidden neurons. Ten different initialisations were tested to find the minimum value for each number of sources. The cost function saturates after five sources and the deviations are due to different random initialisation of the network

worse. The poor signal to noise ratio shows that the data really lies in a nonlinear subspace.

Figure 12 depicts the results after 2000 iterations with nonlinear factor analysis followed by a rotation with a linear FastICA. Now the signal to noise ratio is 13.2 dB and the sources have clearly been retrieved. Figure 13 shows the final result after another 5500 iterations with nonlinear independent factor analysis algorithm. The signal to noise ratio has further improved to 17.3 dB.

It would also be possible to avoid using the nonlinear independent factor analysis algorithm by running first 7500 iterations with linear factor analysis algorithm and then applying the linear ICA. The disadvantage would be that the cost function would not take into account the non-Gaussianity. The signal to noise ratio after 7500 iterations with linear factor analysis algorithm followed by the linear ICA was 14.9 dB, which shows that taking the non-Gaussianity into account during estimation of the nonlinear mapping helps the nonlinear independent factor analysis algorithm to find better estimates for the sources.

5.4 Process Data

This data set consists of 30 time series of length 2480 measured from different sensors from an industrial pulp process. An expert has preprocessed the signals by roughly compensating for time lags of the process which originate from the finite speed of pulp flow through the process.

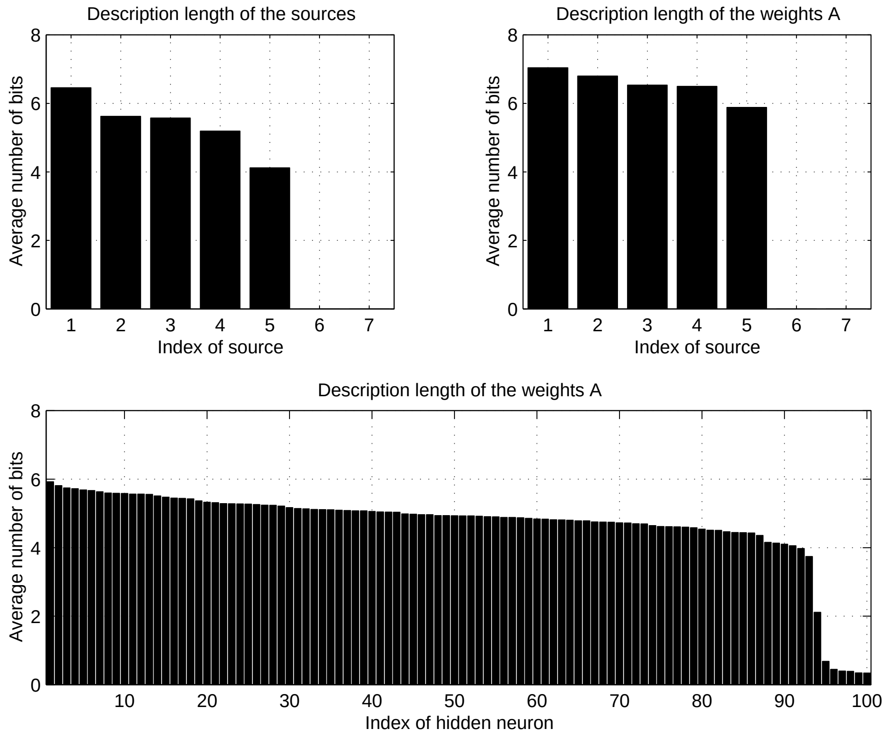

Fig. 10. The network is able to prune away unused parts. This can be monitored by measuring the description length of different variables. The sources and hidden neurons are sorted by decreasing description length

In order to get an idea of the dimensionality of the data, linear factor analysis was applied to the data. The result is shown in Fig. 14. The same figure shows also the results with nonlinear factor analysis. It appears that the data is quite nonlinear since the nonlinear factor analysis is able to explain as much data with 10 components as the linear factor analysis with 21 components.

Several different numbers of hidden neurons and sources where tested with different random initialisations with nonlinear factor analysis and it turned out that the cost function was minimised for a network having 10 sources and 30 hidden neurons. The same network was chosen for nonlinear independent factor analysis, i.e., after 2000 iterations with linear factor analysis the sources were rotated with FastICA and each source was modelled with a mixture of three Gaussian distributions. The resulting sources are shown in Fig. 15.

Figure 16 shows the 30 original time series of the data set, one time series per plot, and in the same plots below the original time series are the reconstructions made by the network, i.e., the posterior means of the output of the network when the inputs were the estimated sources shown in Fig. 15. The original signals shown

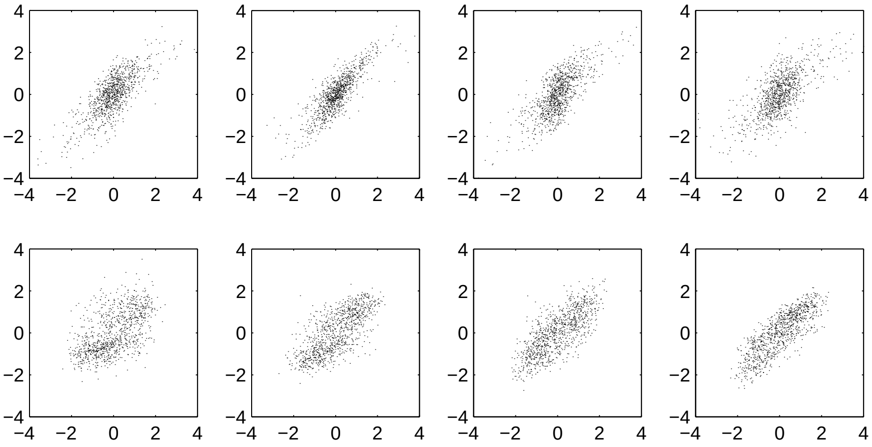

Fig. 11. Original sources are on the x-axis of each scatter plot and the sources estimated by a linear ICA are on the y-axis. Signal to noise ratio is 0.7 dB

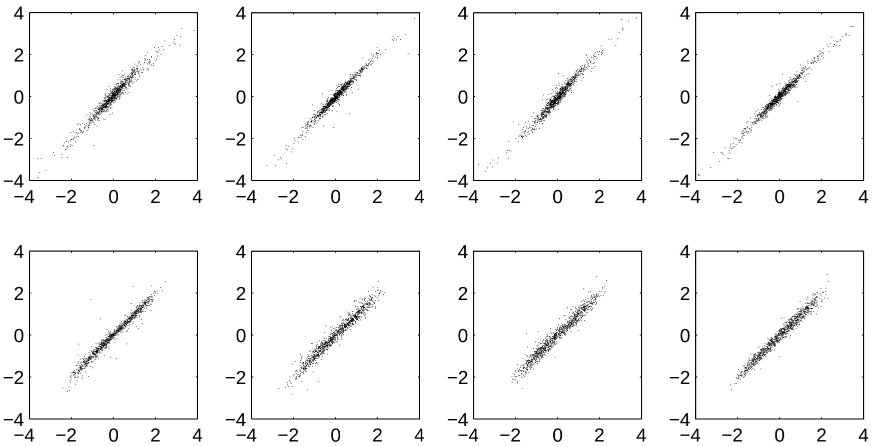

Fig. 12. Scatter plots of the sources after 2000 iterations of nonlinear factor analysis followed by a rotation with a linear ICA. Signal to noise ratio is 13.2 dB

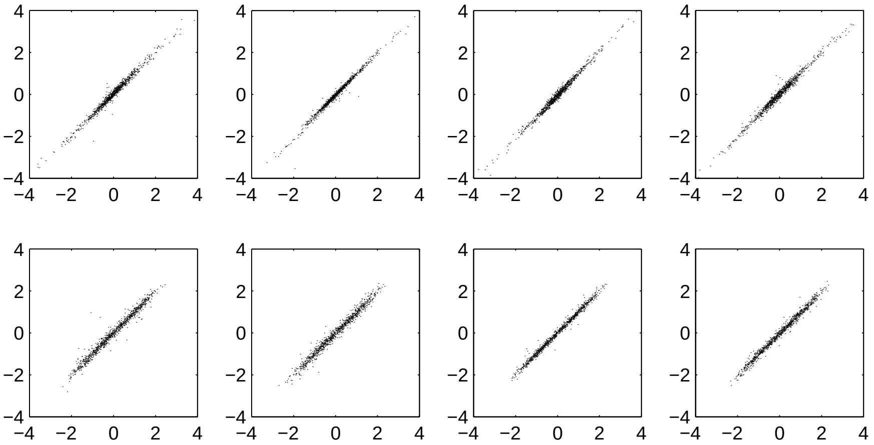

Fig. 13. The network in Fig. 12 has been further trained for 5500 iterations with nonlinear independent factor analysis. Signal to noise ratio is 17.3 dB

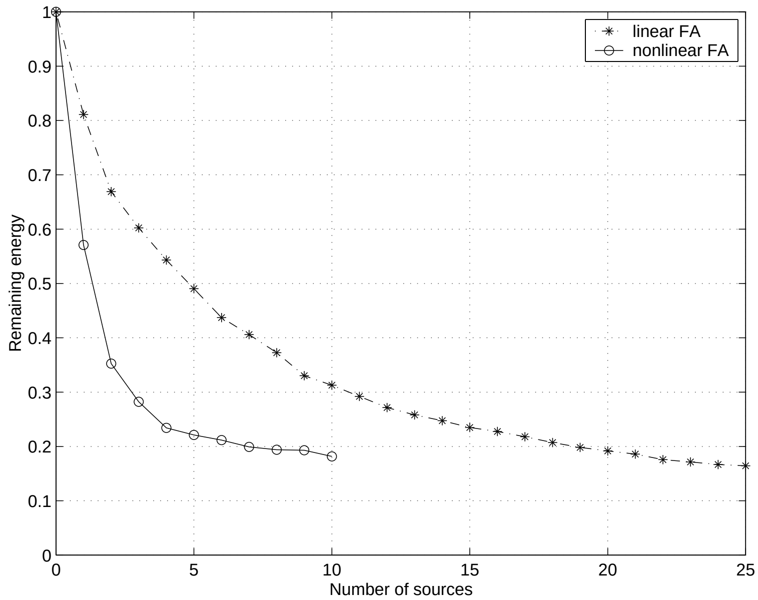

Fig. 14. The graph shows the remaining energy in the process data as a function of the number of extracted components in linear and nonlinear factor analysis

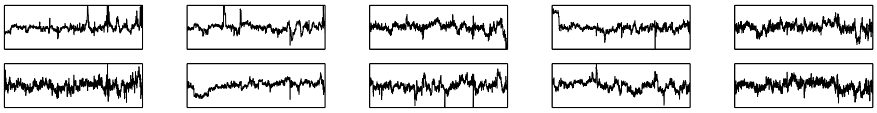

Fig. 15. The ten estimated sources from the industrial pulp process. Time increases from left to right

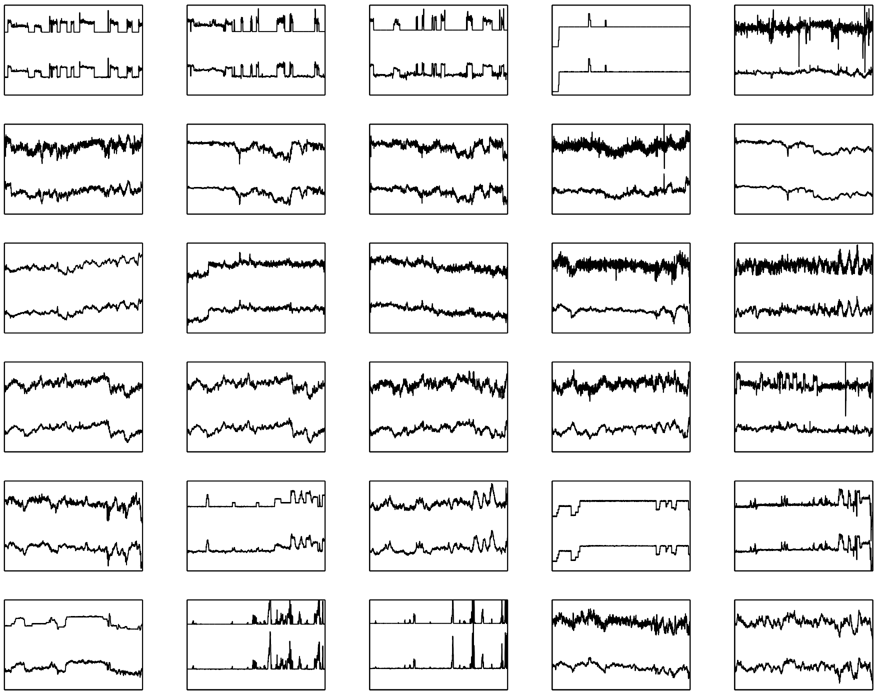

Fig. 16. The 30 original time series are shown on each plot on top of the reconstruction made from the sources shown in Fig. 15

great variability but the reconstructions are strikingly accurate. In some cases it even seems that the reconstruction is less noisy than the original signal. This is somewhat surprising since the time dependencies in the signal were not included in the model. The observation vectors could be arbitrarily shuffled and the model would still give the same result.

Initial studies are pointing to the direction that the estimated source signals can have meaningful physical interpretations. The results are encouraging but further studies are needed to verify the interpretations of the signals.

6 Comparison with Existing Methods

The idea of representing the data with a nonlinear coordinate system is by no means new and several algorithms for learning the coordinate system have been proposed.

6.1 SOM and GTM

Self-organising maps (SOM) [5] and generative topographic mapping (GTM) [2] define a nonlinear coordinate system by stating the coordinates of lattice points called model vectors. The methods are in wide use, particularly the computationally efficient SOM, but the dimension of the latent space is normally quite small. Two-dimensional latent space is the most typical one because it allows an easy visualisation for human users.

The disadvantage of SOM and GTM is that the number of parameters required to describe the mapping from latent variables to observations grows exponentially with the dimension of the latent space. Our main motivation for using MLP network as the nonlinear mapping is that its parametrisation scales linearly with the dimension of the latent space. In this respect the mapping of MLP network is much closer to a linear mapping which has been proven to be applicable for very high dimensional latent spaces. SOM and GTM would probably be better models for the helical data set in Sect. 5.2, but the rest of the experiments have latent spaces whose dimensions are so large that SOM or GTM models would need very many parameters.

6.2 Auto-Associative MLPs

Auto-associative MLP networks have been used for learning similar mappings as we have done. Both the generative model and its inversion are learned simultaneously, but separately without utilising the fact that the models are connected. This means that the learning is much slower than in this case where the inversion is defined as a gradient descent process.

Much of the work with auto-associative MLPs uses point estimates for weights and sources. As argued in the beginning of the chapter, it is then impossible to reliably choose the structure of the model and problems with over- or underlearning may be severe. Hochreiter and Schmidhuber have used and MDL based method which does estimate the distribution of the weights but has no model for the sources [3]. It is then impossible to measure the description length of the sources.

6.3 Generative Learning with MLPs

MacKay and Gibbs briefly report using stochastic approximation to learn a generative MLP network which they called a density network because the model defines a density of the observations [6]. Although the results are encouraging, they do not prove the advantages of the method over SOM or GTM because the model is very simple; noise level is not estimated from the observations and the latent space had only two dimensions. The computational complexity of the method is significantly greater than in the parametric approximation of the posterior presented here, but it might be possible to combine the methods by finding initial approximation of the posterior probability with parametric approximation and then refining it with more elaborate stochastic approximation.

In [7], a generative MLP network was optimised by gradient based learning. The cost function was reconstruction error of the data and a point estimate was used for all the unknown variables. As argued in Sect. 2, this means that it is not possible to optimise model structure and the method is prone to overfitting.

7 Discussion

7.1 Validity of the Approximations

The posterior pdf of all the unknown variables was approximated with a Gaussian density with diagonal covariance, which means that the variables were assumed independent given the observations. The Gaussianity assumption is not severe since the parametrisation is chosen so as to make the posterior close to Gaussian. If the hidden neurons were linear, the posterior of the sources, weights and biases would, in fact, be exactly Gaussian. Gaussian approximation therefore penalises strong nonlinearities to some extent.

The posterior independence seems to be the most unrealistic assumption. It is probable that a change in one of the weights can be compensated by changing the values of other weights and sources, which means that they have posterior dependencies.

In general, the cost function tries to make the approximation of the posterior more accurate, which means that during learning the posterior will also try to be more independent. In PCA, the mapping has a degeneracy which will be used by the algorithm to do exactly this. In linear PCA the mapping is such that the sources are independent a posteriori. In the nonlinear factor analysis, the dependencies of the sources are different in different parts of the latent space and it would be reasonable to model these dependencies. Computational load would not increase significantly since the Jacobian matrix computed in the algorithm can be used also for estimating the posterior interdependencies of the sources. For the sake of simplicity, the derivations were not included here.

It should be possible to do the same for the nonlinear independent factor analysis, but it would probably be necessary to assume the different Gaussians of each source to be independent. Otherwise the posterior approximation of $Q(\mathrm{M}|\mathrm{X})$ would be computationally too intensive.

The other approximation was done when approximating the nonlinearities of the hidden neurons by Taylor's series expansions. For small variances this is valid and it is therefore good to check that the variances of the inputs for the hidden neurons are not outside the range where the approximation is valid. In the computation of the gradients, some terms were neglected to discourage the network from adapting itself to areas of parameter space where the approximation is inaccurate. Experiments have proven that this seems to be working. For the network which minimised the cost function in Fig. 8, for instance, the maximum variance of the input for a hidden neurons was 0.06. Even this maximum value is safely below the values where the approximation could become too inaccurate.

7.2 Initial Inversion by Auxiliary MLP

During learning, both the sources and the mapping of the network evolve together. When the network is presented new data, it is necessary to find the estimates of the sources corresponding to the new data. This can be difficult using similar update process as was used in learning because it is possible that during learning the network develops local minima which make later inversion difficult.

Experiments have shown that it is possible to learn an auxiliary MLP network which will estimate the mapping from observations to sources and can thus be used to initialise the sources given new data. The resulting system with two MLP networks resembles auto-associative MLP network. As was argued before, learning only the generative model is faster than learning a deeper auto-associative MLP with both the generative model and its inverse. Initial experiments have also shown that updates of the sources after the initialisation with the auxiliary MLP network lead to better estimates of the sources.

7.3 Future Directions

In principle, both the nonlinear factor analysis and independent factor analysis can model any time-independent distribution of the observations. MLP networks are universal approximators for mappings and mixture-of-Gaussians for densities. This does not mean, however, that the models described here would be optimal for any time-independent data sets, but the Bayesian methods which were used in the derivation of the algorithms allow easy extensions to more complicated models. It is also easy to use Bayesian model comparison to decide with model is most suited for the data set at hand.

An important extension would be the modelling of dependencies between consecutive sources $\boldsymbol{s}(t)$ and $\boldsymbol{s}(t+1)$ because many natural data sets are time series. For instance both the speech and process data sets used in the experiments clearly have strong time-dependencies.

In the Bayesian framework, treatment of missing values is simple which opens up interesting possibilities for the nonlinear models described here. A typical pattern recognition task can often be divided in unsupervised feature extraction and supervised recognition phases. Using the proposed method, the MLP network can be used for both phases. The data set for the unsupervised feature extraction would have only the raw data and the classifications would be missing. The data set for supervised phase would include both the raw data and the desired outputs and the network. From the point of view of the method presented here, there is no need to make a clear distinction between unsupervised and supervised learning phases as any data vectors can have any combination of missing values. The network will model the joint distribution of all the observations and it is not necessary to specify which of the variables will be the classifications and which are the raw data.

Acknowledgements

The authors wish to thank M.Sc. Xavier Giannakopoulos for his help with the simulations with the nonlinear independent factor analysis and M.Sc. Esa Alhoniemi for his help with the pulp process data.

References

1. Hagai Attias. Independent factor analysis. *Neural Computation*, 11(4):803–851, 1999.
2. Christopher M. Bishop, Markus Svensén, and Christopher K. I. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998.
3. Sepp Hochreiter and Jürgen Schmidhuber. LOCOCODE performs nonlinear ICA without knowing the number of sources. In *Proceedings of the ICA'99*, pages 149–154, Aussois, France, 1999.
4. A. Hyvärinen and E. Oja. A fast fixed-point algorithm for independent component analysis. *Neural Computation*, 9(7):1483–1492, 1997.
5. Teuvo Kohonen. *Self-Organizing Maps*. Springer, 1995.
6. David J. C. MacKay and Mark N. Gibbs. Density networks. In Jim Kay, editor, *Proceedings of Society for General Microbiology Edinburgh meeting*, 1997.
7. Jong-Hoon Oh and H. Sebastian Seung. Learning generative models with the up-propagation algorithm. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 605–611. MIT Press, 1998.