BUILDING BLOCKS FOR HIERARCHICAL LATENT VARIABLE MODELS

Harri Valpola, Tapani Raiko and Juha Karhunen

Helsinki University of Technology, Neural Networks Research Centre
P.O.Box 5400, FIN-02015 HUT, Espoo, Finland
E-mail: `{Harri.Valpola, Tapani.Raiko, Juha.Karhunen}@hut.fi`
URL: `http://www.cis.hut.fi/projects/ica/bayes/`

ABSTRACT

We introduce building blocks from which a large variety of latent variable models can be built. The blocks include continuous and discrete variables, summation, addition, nonlinearity and switching. Ensemble learning provides a cost function which can be used for updating the variables as well as optimising the model structure. The blocks are designed to fit together and to yield efficient update rules. Emphasis is on local computation which results in linear computational complexity. We propose and test a structure with a hierachical nonlinear model for variances and means.

1. INTRODUCTION

We report principles which have been found useful in designing and learning large factor-analysis-like latent variable models. The design is based on a small number of basic building blocks which can be flexibly combined. Three important issues arise within this design: 1) the need for a cost function which can be used for learning the model structure, 2) a learning method which avoids over-fitting and 3) the requirement of roughly linear computational complexity for scalability.

Ensemble learning [1] has proven to satisfy these requirements. 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], nonlinear units [3, 4] and MLP networks to model nonlinear observation mappings [5] and nonlinear dynamics of the sources [6]. Ensemble learning has also been applied to large discrete models such as belief networks and hidden Markov models.

In this paper we discuss models which are build from addition and multiplication, Gaussian variables possibly followed by a nonlinearity, as well as discrete variables and switching units. Various model structures proposed in the literature can be build out of these elements and we also present some new model structures. They utilise Gaussian variables which model the variance of other Gaussian variables allowing the variance to have a hierarchical or dynamical model. Related model structures have been proposed for instance in [7, 8, 9, 10, 11, 12] but with these methods it is difficult to learn the structure of the model or compare different model structures.

This paper is organised as follows. Section 2 gives a brief overview of ensemble learning. The building blocks are introduced in Section 3 and various models structures which utilise them are discussed in Section 4. Experiments with a hierarchical nonlinear model for means and variances are reported in Section 5.

This research has been funded by the European Commission project BLISS, and the Finnish Center of Excellence Programme (2000 - 2005) under the project New Information Processing Principles.

2. ENSEMBLE LEARNING

This section gives a brief overview of ensemble learning with emphasis on solutions yielding linear computational complexity. Thorough introductions to ensemble learning can be found for instance in [13, 14].

Ensemble learning is a method for approximating posterior probability distributions. It enables to choose a posterior approximation ranging from point estimates to exact posterior. The misfit of the approximation is measured by the Kullback-Leibler divergence between the posterior and its approximation. Let us denote the observed variables by $\mathbf{X}$, the latent variables (parameters) of the model by $\boldsymbol{\theta}$ and the approximation of the true posterior $p(\boldsymbol{\theta} \mid \mathbf{X})$ by $q(\boldsymbol{\theta})$. The cost function C used in ensemble learning is

$$C = \left\langle \ln \frac{q(\boldsymbol{\theta})}{p(\mathbf{X}, \boldsymbol{\theta})} \right\rangle = \left\langle \ln \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta} \mid \mathbf{X})} \right\rangle - \ln p(\mathbf{X}) \, , \quad (1)$$

where the operator $\langle \cdot \rangle$ denotes an expectation over the distribution $q(\boldsymbol{\theta})$. Note that for practical reasons the cost function equals the Kullback-Leibler divergence only up to a constant $-\ln p(\mathbf{X})$. This means that the cost function can be turned into a lower bound of the model evidence $p(\mathbf{X})$ which can then be used for learning the model structure.

2.1. Linear computational complexity

Each variable in the model yields one multiplicative term in $p(\mathbf{X}, \boldsymbol{\theta})$. The terms can be expressed in the form $p(\text{variable} \mid \text{parents})$. The parents can be either computational nodes, such as summation, addition or switch, or other variables. The difficult part of the cost function is $\langle \ln p(\mathbf{X}, \boldsymbol{\theta}) \rangle$ which is taken over $q(\boldsymbol{\theta})$. The logarithm splits the product of simple terms into a sum. If each of the simple terms can be computed in constant time, the overall computational complexity is linear.

In general, the computation time is constant if the parents are independent 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 $q(\boldsymbol{\theta})$.

According to our experience, almost maximally factorial $q(\boldsymbol{\theta})$ suffice for latent variable models. It seems that a good model structure is usually more important than a good approximation of the posterior probability of the model. Density estimates of continuous valued latent variables offer a large 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 are almost as efficient as point estimates.

2.2. Pruning and local minima

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 and 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 [6] 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 axis 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.

3. BUILDING BLOCKS

In this section we introduce the building blocks and equations for computation with them. The building blocks consist of variable nodes and computation nodes. The symbols we use for them are shown in Figure 1. We shall refer to inputs and outputs of the nodes. For variable nodes, input means a value which is used for the prior distribution and output is the value of the variable. For computation nodes, output is a fixed function of the inputs.

The variable nodes can be either continuous valued with Gaussian prior models or discrete with soft-max prior models. Gaussian and soft-max are chosen because the outputs of the Gaussian nodes can be used as inputs to Gaussian or soft-max nodes as will be explained shortly. This makes the nodes compatible with each other. Each variable can be either observed or latent.

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. Consequently the cost function can be minimised based on gradients w.r.t. these expectations computed by back-propagation.

The input for prior mean of a Gaussian node requires the mean and variance. With a suitable parametrisation, mean and expected exponential are required from the input for prior variance. The output of a Gaussian node can provide the mean, variance and expected exponential and can thus be used as an input to both the mean and variance of another Gaussian node. Gaussian nodes are suitable parents for discrete nodes as well since soft-max requires the mean and expected exponential of the input. The expectations required by the inputs and provided by the outputs of different nodes are listed below:

Output provides:

Gaussian	$\langle\cdot\rangle$	$\mathrm{Var}\{\cdot\}$	$\langle\exp\cdot\rangle$
Gaussian with nonlinearity	$\langle\cdot\rangle$	$\mathrm{Var}\{\cdot\}$	
addition	$\langle\cdot\rangle$	$\mathrm{Var}\{\cdot\}$	$\langle\exp\cdot\rangle$
multiplication	$\langle\cdot\rangle$	$\mathrm{Var}\{\cdot\}$	
switch	$\langle\cdot\rangle$	$\mathrm{Var}\{\cdot\}$	$\langle\exp\cdot\rangle$

Prior for variable nodes requires:

mean of Gaussians	$\langle\cdot\rangle$	$\mathrm{Var}\{\cdot\}$	
variance of Gaussians	$\langle\cdot\rangle$		$\langle\exp\cdot\rangle$
soft-max of discrete	$\langle\cdot\rangle$		$\langle\exp\cdot\rangle$

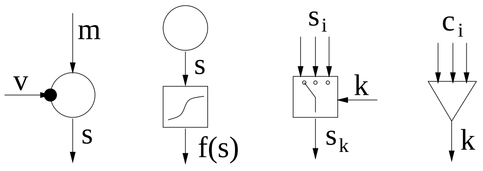

Fig. 1. First from left: A Gaussian latent variable s, marked with a circle, has a prior mean m and a prior variance $\exp(-v)$. Second: A nonlinearity f is applied immediately after a Gaussian variable. Third: A switch selects the kth continuous valued input as the output. Fourth: Discrete variable k, marked with a triangle, has a soft-max prior derived from continuous valued variables c_i.

3.1. Gaussian variables

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

$$C_{s,p} = \frac{1}{2}\Big\{ \langle \exp v\rangle \Big[(\langle s\rangle - \langle m\rangle)^2 + \mathrm{Var}\{m\} + \\ + \mathrm{Var}\{s\} \Big] - \langle v\rangle + \ln 2\pi \Big\}. \quad (2)$$

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 $\widetilde{s}$: $q(s) = N(s;\overline{s},\widetilde{s})$. This yields

$$C_{s,q} = -\frac{1}{2}\ln 2\pi e\widetilde{s} \quad (3)$$

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 can be shown to be $\langle \exp s\rangle = \exp(\overline{s} + \widetilde{s}/2)$. The outputs of observed nodes are scalar values instead of distributions and thus $\langle s\rangle = s$, $\mathrm{Var}\{s\} = 0$ and $\langle \exp s\rangle = \exp s$.

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 $b[(\overline{s} - a)^2 + \widetilde{s}] + c\langle \exp s\rangle$, where $\partial C_p/\partial \overline{s} = 2b(\overline{s} - a)$, $\partial C_p/\partial \widetilde{s} = b$ and $\partial C/\partial \langle \exp s\rangle = c$. This assumption holds exactly if the output of the node is propagated to Gaussian nodes only and not to discrete nodes. If the output is used by a discrete node with a soft-max prior, this term gives an upper bound of C_p as will be explained later. 3) Third, the minimum of $C = C_p + C_q$ is solved. This can be done analytically if $c = 0$, otherwise the minimum is obtained iteratively.

3.2. Addition and Multiplication

Addition and multiplication nodes can be used e.g. for constructing linear mappings and affine transformations between the variables. Denoting the inputs by s_i, the outputs are $\sum_i s_i$ for addition and $\prod_i s_i$ for multiplication nodes. The mean, variance and expected exponential of the addition node are

$$\begin{aligned} \langle s_1 + s_2\rangle &= \langle s_1\rangle + \langle s_2\rangle & (4)\\ \mathrm{Var}\{s_1 + s_2\} &= \mathrm{Var}\{s_1\} + \mathrm{Var}\{s_2\} & (5)\\ \langle \exp(s_1 + s_2)\rangle &= \langle \exp s_1\rangle \langle \exp s_2\rangle & (6) \end{aligned}$$

assuming s_i independent. For multiplication node the except exponential cannot be evaluated without knowing the exact distribution of the inputs. Assuming independence between s_i, the mean and the variance of the output are

$$\begin{aligned} \langle s_1 s_2\rangle &= \langle s_1\rangle \langle s_2\rangle & (7)\\ \mathrm{Var}\{s_1 s_2\} &= \langle s_1\rangle^2 \mathrm{Var}\{s_2\} & (8)\\ &\quad + \mathrm{Var}\{s_1\}\Big(\langle s_2\rangle^2 + \mathrm{Var}\{s_2\}\Big). \end{aligned}$$

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

3.3. Gaussian variable with nonlinearity

A nonlinear computation node can be used for constructing nonlinear mappings between the variable nodes. For most nonlinear functions it is impossible to compute the required expectations analytically, but for the function $f(s) = \exp(-s^2)$ the mean and variance have analytical expressions provided that they have Gaussian inputs, i.e. the nonlinearity has to follow immediately after a Gaussian node [15]. The required expectations are

$$\begin{aligned} \langle f(s)\rangle &= \exp\left(-\frac{\overline{s}^2}{2\widetilde{s} + 1}\right)(2\widetilde{s} + 1)^{-\frac{1}{2}} & (9)\\ \left\langle f(s)^2\right\rangle &= \exp\left(-\frac{2\overline{s}^2}{4\widetilde{s} + 1}\right)(4\widetilde{s} + 1)^{-\frac{1}{2}}. & (10) \end{aligned}$$

The variance is obtained by $\mathrm{Var}\{f(s)\} = \left\langle f^2(s)\right\rangle - \langle f(s)\rangle^2$. The update of a Gaussian node followed by the nonlinearity is similar to the plain Gaussian node: 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. This assumption holds since the nonlinearity can only propagate to the mean of Gaussian nodes.

3.4. Discrete variables

The prior probabilities of discrete variables with n possible values can be assigned from n continuous valued signals c_i using soft-max prior:

$$p(k = i \mid \mathbf{c}) = \frac{\exp c_i}{\sum_{j=1}^{n} \exp c_j}. \tag{11}$$

The term $\langle -\ln p(k \mid \mathbf{c})\rangle$ of the cost function cannot be computed exactly but it can be approximated from above by using

$$\langle -\ln p(k \mid \mathbf{c})\rangle = \left\langle -c_k + \ln \sum_{j=1}^{n} \exp c_j \right\rangle \tag{12}$$

$$\leq -\langle c_k\rangle + \ln \sum_{j=1}^{n} \langle \exp c_j\rangle = C_{k,p}\,, \tag{13}$$

which follows from the Jensens' inequality assuming all the inputs independent. Note that the terms $\langle \exp c_j\rangle$ appear inside the concave logarithmic function. A linear approximation based on the derivative w.r.t. $\langle \exp c_j\rangle$ therefore yields an upper bound for the cost function.

For latent discrete variables there are no restrictions to the posterior approximation $q(k)$. The term $C_{k,q}$ in the cost function arising from $\langle \ln q(k)\rangle$ is simply the negative entropy of $q(k)$.

The update of $q(k)$ is analogous to Gaussian variables: the gradient of C_p w.r.t. the vector $q(k = i)$ with $1 \leq i \leq n$ is assumed to arise from a linear term $\sum_i q(k = i) C_p(k = i)$, where $C_p(k = i)$ denotes the value of C_p assuming that $q(k = i) = 1$. The linearity assumption holds exactly if the value of the discrete node propagates only to Gaussian variables (through switches) and corresponds to an upper bound of the cost function if the values are used by other discrete variables with soft-max prior. It can be shown that at the minimum of the cost function it holds $q(k = i) \propto \exp(-C_p(k = i))$.

3.5. Switch

In a switch node, an input k with n discrete values selects one of the n continuous valued inputs s_i as the output: $s_{\text{out}} = s_k$. The required expectations are as follows:

$$\langle s_{\text{out}}\rangle = \sum_{i=1}^{n} q(k = i)\,\langle s_i\rangle \tag{14}$$

$$\langle s_{\text{out}}^2\rangle = \sum_{i=1}^{n} q(k = i)\,\langle s_i^2\rangle \tag{15}$$

$$\langle \exp s_{\text{out}}\rangle = \sum_{i=1}^{n} q(k = i)\,\langle \exp s_i\rangle\,. \tag{16}$$

The variance is obtained by $\operatorname{Var}\{s_{\text{out}}\} = \langle s_{\text{out}}^2\rangle - \langle s_{\text{out}}\rangle^2$ and $\langle s_i^2\rangle = \langle s_i\rangle^2 + \operatorname{Var}\{s_i\}$.

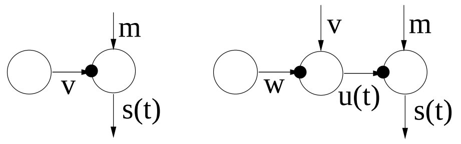

Fig. 2. Left: Source $s(t)$ has a time independent prior variance v. Right: A variance neuron is included to give a time dependent prior variance $u(t)$ for the source $s(t)$.

4. EXAMPLE STRUCTURES

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; 2) nonlinearity is always immediately after a Gaussian latent variable; 3) outputs of multiplication or nonlinearity cannot propagate to soft-max or variance prior because the expected exponential cannot be evaluated; 4) the output of a discrete latent variable can only be used for a switch and 5) there should be only one computational path from a latent variable to variable. If there are multiple paths, ensemble learning becomes more complicated [5] and the situation is out of the scope of this paper.

4.1. 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. We propose a variance neuron shown in Figure 2. 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. This can be useful for instance in modelling outliers in the observations.

4.2. Linear Independent Factor Analysis

The addition and multiplication nodes can be used for building an affine transformation from Gaussian source nodes $\mathbf{s}(t)$ to Gaussian observation nodes $\mathbf{x}(t)$. This corresponds to linear factor analysis. With an independent mixture-of-Gaussians prior for each of the sources, the model corresponds to linear independent factor analysis [2]. Figure 3 shows how switches can be used to build such a prior for a source $s(t)$. A variance neuron is used in order to prevent multiple paths from the discrete node to the source.

4.3. Hierarchical Nonlinear Variance Model

Figure 4 shows the structure for the hierarchical nonlinear variance (HNV) model. It utilises variance neurons and nonlinearities in building a hierarchical model for both the

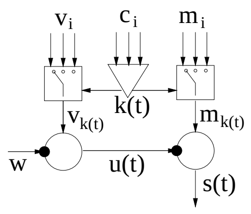

Fig. 3. A mixture-of-Gaussians prior for $s(t)$ is achieved using switches.

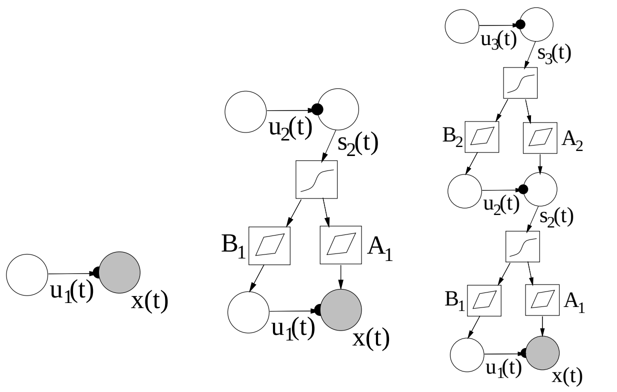

Fig. 4. HNV model can be built up in stages. Left: A variance neuron is attached to each Gaussian observation node. The nodes represent vectors. Middle: A layer of sources with variance neurons attached to them is added. The nodes next to the weight matrices $\mathbf{A}_1$ and $\mathbf{B}_1$ represent 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.

means and variances. Without the variance neurons the model would correspond to a multi-layer perceptron with latent variables at hidden neurons. Note that computation nodes as hidden neurons would result in multiple paths from upper layer latent variables to the observations. This type of structure was used in [5] and it has a quadratic as opposed to linear computational complexity.

4.4. Dynamics

From the point of view of the equations for the nodes, connections forward in time are no different from connections down the hierarchy as long as the network remains a directed acyclic graph. The building blocks can therefore be used to build dynamical models by defining mappings from past sources to future sources or observations.

5. THE BARS PROBLEM

We have tested the HNV model to an extension of the bars problem [16]. The data set consists of 6×6 pixel image patches with horizontal and vertical bars. In addition to the regular bars, we used horizontal and vertical variance bars

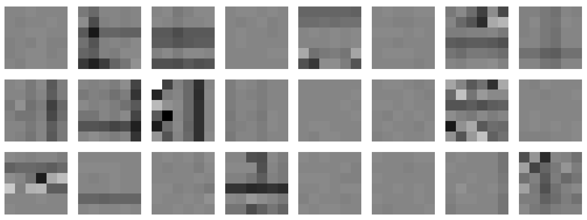

Fig. 5. Samples from the 1000 image patches used in the bars problem.

that are manifested by increased variance. Samples of the image patches are shown in Figure 5.

Data was generated by first choosing whether vertical and/or horizontal orientations are active, each with probability 1/2 independently. If an orientation is active, there is a probability 1/3 for each bar of that orientation 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 was added to each pixel.

5.1. Learning Procedure

The network was initialised in stages shown in Figure 4. The first layer was added after 20 sweeps and the second after 100 sweeps. Each sweep corresponds to updating each latent variable node once. To prevent falling to local minima, 1) automatic pruning was discouraged initially, 2) new sources were generated and pruned sources removed from time to time and 3) the activations of the sources were reset a few times.

5.2. Results

In the initial phases of learning, some of the sources represented multiple bars and there were multiple sources representing a single bar. There was also a source which was specialised to diminish variance in cases where both the horizontal and vertical orientations were inactive. These local minima were escaped, however, as the weights after 1200 sweeps in Figure 6 demonstrate.

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. The two sources on the second layer correspond to the horizontal and vertical orientations and the 18 sources on the first 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 $\mathbf{A}_2$.

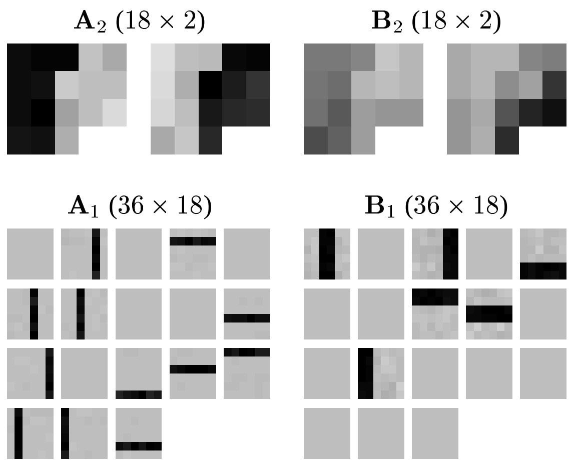

Fig. 6. Posterior means of the weight matrices after 1200 sweeps. The matrices are organised in patches and dark shades represent positive values.

6. DISCUSSION

The building blocks discussed in this paper can be used for constructing a wide variety of models. An important future line of research will be 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. In the experiments reported here, we did not make use of sparsely connected networks but for large models they are likely to prove useful.

The hierarchical nonlinear variance model was shown to be able to learn the structure of the underlying data generating process. In most real cases the generative process can be expected to be more complex, but the same model structure can handle a variety of different cases. In some of the preliminary experiments we have conducted on image data, second-level sources which resemble complex cells [11, 12] have emerged. However, so far we have used fully connected mappings which seems to discourage the formation of complex-cell-like sources as each of them typically models the variance of only a small number of the lower-level sources.

Externally the variance neurons appear as any other Gaussian nodes. It is therefore easy to build for instance 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 paper was restricted to models with purely local computation. 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 elements in Figure 3 for instance can be grouped as in [2].

7. REFERENCES

[1] G. E. Hinton and D. van Camp, “Keeping neural networks simple by minimizing the description length of the weights,” in *Proc. COLT’93*, pp. 5–13, 1993.

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

[3] B. J. Frey and G. E. Hinton, “Variational learning in nonlinear gaussian belief networks,” *Neural Computation*, vol. 11, no. 1, pp. 193–214, 1999.

[4] K. P. Murphy, “A variational approximation for Bayesian networks with discrete and continuous latent variables,” In *Proc. UAI–99*, pp. 457–466, 1999.

[5] H. Lappalainen and A. Honkela, “Bayesian nonlinear independent component analysis by multi-layer perceptrons,” in *Advances in Independent Component Analysis* (M. Girolami, ed.), pp. 93–121, Springer-Verlag, 2000.

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

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

[8] J.-F. Cardoso, “Multidimensional independent component analysis,” In *Proc. ICASSP’98*, pp. 1941–1944, 1998.

[9] D.-T. Pham and J.-F. Cardoso, “Blind separation of instantaneous mixtures of non stationary sources,” In *Proc. ICA 2000*, pp. 187–192, 2000.

[10] Z. Ghahramani and G. E. Hinton, “Hierarchical non-linear factor analysis and topographic maps,” In *Adv. in Neur. Inf. Proc. Syst. 10, NIPS*97*, pp. 486–492, 1998.

[11] A. Hyvärinen and P. O. Hoyer, “Emergence of topography and complex cell properties from natural images using extensions of ICA,” In *Adv. in Neur. Inf. Proc. Syst. 12, NIPS*99*, pp. 827–833, 2000.

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

[13] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, “An introduction to variational methods for graphical models,” in *Learning in Graphical Models* (M. I. Jordan, ed.), pp. 105–161, The MIT Press, 1999.

[14] H. Lappalainen and J. W. Miskin, “Ensemble learning,” in *Advances in Independent Component Analysis* (M. Girolami, ed.), pp. 76–92, Springer-Verlag, 2000.

[15] Z. Ghahramani and S. T. Roweis, “Learning nonlinear dynamical systems using an EM algorithm,” In *Adv. in Neur. Inf. Proc. Syst. 11, NIPS*98*, pp. 599–605, 1999.

[16] P. Dayan and R. S. Zemel, “Competition and multiple cause models,” *Neural Computation*, vol. 7, no. 3, pp. 565–579, 1995.