
ELSEVIER

Theoretical Computer Science 287 (2002) 187–207

Theoretical Computer Science

www.elsevier.com/locate/tcs

Unsupervised learning in neural computation

Erkki Oja

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

Abstract

In this article, we consider unsupervised learning from the point of view of applying neural computation on signal and data analysis problems. The article is an introductory survey, concentrating on the main principles and categories of unsupervised learning. In neural computation, there are two classical categories for unsupervised learning methods and models: first, extensions of principal component analysis and factor analysis, and second, learning vector coding or clustering methods that are based on competitive learning. These are covered in this article. The more recent trend in unsupervised learning is to consider this problem in the framework of probabilistic generative models. If it is possible to build and estimate a model that explains the data in terms of some latent variables, key insights may be obtained into the true nature and structure of the data. This approach is also briefly reviewed.

1. Introduction

Unsupervised learning is a deep concept that can be approached from very different perspectives, from psychology and cognitive science to engineering. It is often called "learning without a teacher". This implies that a learning human, animal, or artificial system observes its surroundings and, based on these observations, adapts its behavior without being told to associate given observations to given desired responses (supervised learning) or without even given any hints about the goodness of a given response (reinforcement learning). Usually, the result of unsupervised learning is a new explanation or representation of the observation data, which will then lead to improved future responses or decisions.

In machine learning and artificial intelligence, such a representation is a set of concepts and rules between these concepts, which give a symbolic explanation for the data. In artificial neural networks, the representation may be a clustering of the data,

E-mail address: erkki.oja@hut.fi (E. Oja).

0304-3975/02/$ - see front matter
PII: S0304-3975(02)00160-3

a discrete map, or a continuous lower-dimensional manifold in the vector space of observations, which explains their structure and may reveal their underlying causes.

Unsupervised learning seems to be the basic mechanism for sensory adaptation, e.g. in the visual pathway [4]. On the engineering side, it is a highly powerful and promising approach to some practical data processing problems like data mining and knowledge discovery from very large databases, or new modes of human-computer interactions in which the software adapts to the requirements and habits of the human user by observing her behaviour.

In this article, we consider unsupervised learning from the point of view of applying neural computation on signal and data analysis problems. The article is an introductory survey, concentrating on the main principles and categories of unsupervised learning. It is not possible to review all the methods in detail—for an excellent collection of articles, see [22]. Neither is it possible to cover the more biological neural modelling approaches in a single paper; for some collections of classical and more recent articles, see e.g. [17,22].

In neural computation, there have been two classical categories for unsupervised learning methods and models: first, extensions of *principal component analysis* (PCA) and *factor analysis* (FA), and second, learning vector coding or clustering methods that are based on competitive learning. These are covered in this article in Sections 3 and 4, respectively.

The more recent trend in unsupervised learning is to consider this problem in the framework of probabilistic generative models. If it is possible to build and estimate a model that explains the data in terms of some latent variables, key insights may be obtained into the true nature and structure of the data. Operations like prediction and compression become easier and rigorously justifiable. This approach is reviewed in Section 5.

2. Unsupervised learning in artificial neural networks

2.1. Supervised vs. unsupervised learning

The starting point for learning in neural networks is a training set of *numerical data vectors*, typically high dimensional. Most of the recent neural network research has focused on networks based on *supervised learning*, like the multi-layer perceptron network, the radial basis function network, or the LVQ network [19]. The purpose is to design nonlinear mappings between given *inputs and outputs*, using a database of training samples from both. Examples are pattern recognition, optical character readers, speech recognition, industrial diagnostics, condition monitoring, modelling complex black box systems for control, and time series analysis and forecasting. Inputs are the vectors of observations, and outputs are either classes, coded as binary vectors, or some kind of desired responses given as numerical vectors. Typically, learning is completely data-driven, without any explicit prior information on the nonlinear classification or regression function. This "black box" modelling approach is shown in Fig. 1.

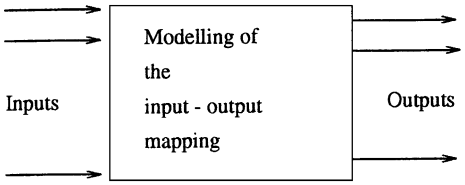

Fig. 1. Black box modelling approach.

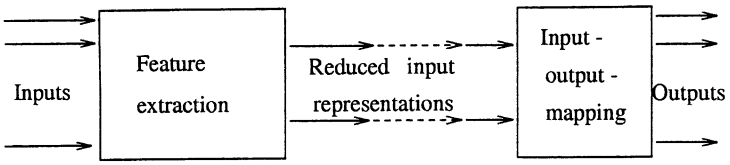

Fig. 2. Feature extraction approach.

In many real world problems like image analysis, data mining, exploratory data analysis, or modelling of large and complex systems, the dimensionality of the input vectors can be very high (of the order of hundreds or even thousands) and the function to be approximated is very nonlinear and complex. Then the networks would require a large number of parameters in order to approximate and generalize well all over the input domain. This means that the amount of training data must grow in proportion to the number of free parameters. Consequently, very large amounts of training data and training time are needed in highly complex problems to form the input–output mappings by the black-box approach. Collecting the training samples would eventually be very expensive if not impossible. This seems to be a major limitation of the supervised learning paradigm for practical data processing problems.

A solution to the problem posed above is to somehow reduce the complexity of the input data. In any meaningful data processing task, the data are not really random but are generated by physical processes or causes of limited complexity [22]. Any decisions on the data, like classification or regression on another set of variables, are actually decisions on these underlying processes or causes. To find the underlying generating variables, *unsupervised learning* is a feasible approach.

Once the observation data are reduced to the underlying variables, it may be much easier to solve the original problem. With a reduced set of input variables, the input–output mapping becomes simpler and less training samples are needed. Splitting the task in two parts also makes it more transparent and allows human expertise to be incorporated especially in the first stage.

For example, in pattern recognition, this approach has always been the standard one. The task is divided in two parts: *feature extraction* which is unsupervised and maps the original input patterns or images to a feature space of reduced dimensions and complexity, followed by classification in this space [46,54]. For designing the classifier, supervised learning is used. This feature extraction approach is shown in Fig. 2.

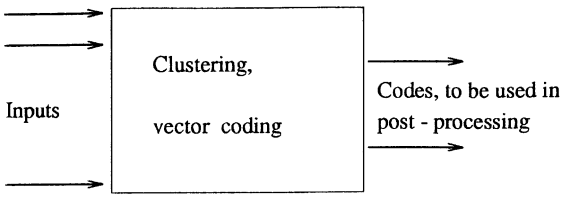

Fig. 3. Clustering approach.

When comparing supervised and unsupervised learning, quite another question is the plausibility of these learning modes in real nervous systems. It may be hard to postulate where the rich supervision signals would come from, for example in sensory adaptation and learning. If we accept the hypothesis that learning is based on synaptic modification, another problem is how supervised learning rules like back-propagation could be implemented locally on the synaptic level. The biological substrate seems to be much more compatible with the unsupervised mode of learning.

2.2. The two basic categories of unsupervised neural learning

The goal of unsupervised learning, finding a new compressed representation for the observations, can be interpreted as coding of the data. Basically, the unsupervised neural learning algorithms fall into one of two categories [19]: first, extensions of the linear *transform coding* methods of statistics, and second, learning *vector coding* methods that are based on competitive learning.

The first class of unsupervised learning methods are motivated by standard statistical methods like PCA or FA, which give a reduced subset of linear combinations of the original input variables. Many of the learning rules for PCA are based on the author's PCA neuron model [35]. A more recent model in this category is that of independent components, which would maximally reduce the redundancy between the latent variables. This leads to the techniques of *independent component analysis* (ICA) and *blind source separation* (BSS) [25]. In the latter technique, a set of parallel time signals such as speech waveforms, electromagnetic measurements from the brain, or financial time series, are assumed to be linear combinations of underlying independent latent variables. The variables, called independent components, are found by efficient ICA learning rules. The approach of basic PCA, FA, and ICA is covered in Section 3.

The second class of methods is close to *clustering*; see Fig. 3. A typical application is *data mining* or profiling from massive databases. It is of interest to find out what kind of typical clusters there are among the data records. In a customer profiling application, finding the clusters from a large customer database means more sharply targeted marketing with less cost. In process modelling, finding the relevant clusters of the process state vector in real operation helps in diagnosis and control. A competitive learning neural network gives an efficient solution to this problem. Section 4 reviews the best-known competitive learning network, the self-organizing map (SOM) introduced by Kohonen [31], and its use in massive data clustering.

3. Principal and independent component analysis

3.1. Principal component analysis

PCA and the closely related Karhunen–Loève Transform, or the Hotelling Transform, as well as FA, are classical techniques in statistical data analysis, feature extraction, and data compression [12,36,54]. Given a set of multivariate measurements, the purpose is to find a smaller set of variables with less redundancy, that would give as good a representation as possible.

The starting point for PCA is a random vector $\mathbf{x}$ with n elements. We may assume that $\mathbf{x}$ is zero mean: $E\{\mathbf{x}\}=0$, with E denoting the expectation with respect to the (unknown) density of $\mathbf{x}$. This can always be achieved by centering $\mathbf{x}$, or computing its mean from an available sample $\mathbf{x}(1),\ldots,\mathbf{x}(T)$ and then subtracting the mean.

Consider a linear combination

$$y_1=\sum_{k=1}^{n} w_{k1}x_k=\mathbf{w}_1^{\mathrm{T}}\mathbf{x}$$

of the elements $x_1,\ldots,x_n$ of the vector $\mathbf{x}$. There $w_{11},\ldots,w_{n1}$ are scalar coefficients or weights, elements of an n-dimensional vector $\mathbf{w}_1$, and $\mathbf{w}_1^{\mathrm{T}}$ denotes the transpose of $\mathbf{w}_1$. We seek for a weight vector $\mathbf{w}_1$ maximizing the PCA criterion

$$J_1^{PCA}(\mathbf{w}_1)=E\{y_1^2\} \tag{1}$$

$$=E\{(\mathbf{w}_1^{\mathrm{T}}\mathbf{x})^2\}=\mathbf{w}_1^{\mathrm{T}}E\{\mathbf{x}\mathbf{x}^{\mathrm{T}}\}\mathbf{w}_1=\mathbf{w}_1^{\mathrm{T}}\mathbf{C}_{\mathbf{x}}\mathbf{w}_1, \tag{2}$$

$$\|\mathbf{w}_1\|=1. \tag{3}$$

The matrix $\mathbf{C}_{\mathbf{x}}$ in Eq. (1) is the $n\times n$ covariance matrix defined for the zero-mean vector $\mathbf{x}$ by

$$\mathbf{C}_{\mathbf{x}}=E\{\mathbf{x}\mathbf{x}^{\mathrm{T}}\}. \tag{4}$$

It is well known from basic linear algebra (see, e.g., [36,12]) that the solution to the PCA problem is given in terms of the unit-length eigenvectors $\mathbf{e}_1,\ldots,\mathbf{e}_n$ of the matrix $\mathbf{C}_{\mathbf{x}}$. The ordering of the eigenvectors is such that the corresponding eigenvalues $d_1,\ldots,d_n$ satisfy $d_1\geqslant d_2\geqslant\cdots\geqslant d_n$. The solution maximizing (1) is given by

$$\mathbf{w}_1=\mathbf{e}_1.$$

Thus the first principal component of $\mathbf{x}$ is $y_1=\mathbf{e}_1^{\mathrm{T}}\mathbf{x}$.

The criterion J_1^{PCA} in Eq. (1) can be generalized to m principal components, with m any number between 1 and n. Then the variances of $y_i=\mathbf{w}_i^{\mathrm{T}}\mathbf{x}$ are maximized under the constraint that the principal component vectors are orthonormal, or $\mathbf{w}_i^{\mathrm{T}}\mathbf{w}_j=\delta_{ij}$ for all $j<i$. The solution is given by the ith eigenvector $\mathbf{w}_i=\mathbf{e}_i$.

Another, alternative criterion for PCA is minimizing the mean square error

$$J_{MSE}^{PCA} = E\left\{\left|\left|\mathbf{x} - \sum_{i=1}^{m} (\mathbf{w}_i^{\mathrm{T}}\mathbf{x})\mathbf{w}_i\right|\right|^2\right\}. \tag{5}$$

assuming that the basis vectors $\mathbf{w}_i$ are orthonormal. The sum in Eq. (5) is the orthogonal projection of $\mathbf{x}$ on the subspace spanned by the vectors $\mathbf{w}_1,\ldots,\mathbf{w}_m$. A minimum (although not the only one) of this criterion is again given by $\mathbf{w}_i = \mathbf{e}_i$; for more details, see [12,25].

To use the closed form solution $\mathbf{w}_i = \mathbf{e}_i$ given above for the PCA basis vectors, the eigenvectors of the covariance matrix $\mathbf{C}_\mathbf{x}$ must be known. In the conventional use of PCA, there is a sufficiently large sample of vectors $\mathbf{x}(t)$ available, from which the mean and the covariance matrix $\mathbf{C}_\mathbf{x}$ can be estimated by standard methods. There are several efficient numerical methods available for solving the eigenvectors of $\mathbf{C}_\mathbf{x}$, e.g. the QR algorithm with its variants [12].

3.2. PCA and neural networks

It is not always feasible to solve the eigenvectors by standard numerical methods. In an on-line data compression application like image or speech coding, the data samples $\mathbf{x}(t)$ arrive at high speed, and it may not be possible to estimate the covariance matrix and solve the eigenvector–eigenvalue problem once and for all.

An alternative is to derive gradient ascent algorithms or other online methods for the maximization problems above. The algorithms will then converge to the solutions of the problems, that is, to the eigenvectors. The advantage of this approach is that such algorithms work on-line, using each input vector $\mathbf{x}(t)$ once as it becomes available and making an incremental change to the eigenvector estimates, without computing the covariance matrix at all. This approach is the basis of the PCA neural network learning rules.

In the *Constrained Hebbian learning rule* introduced by the author [35], the gradient of y_1^2 is taken with respect to $\mathbf{w}_1$ and the normalizing constraint $\|\mathbf{w}_1\| = 1$ is taken into account. The learning rule is

$$\mathbf{w}_1(t+1) = \mathbf{w}_1(t) + \gamma(t)[y_1(t)\mathbf{x}(t) - y_1^2(t)\mathbf{w}_1(t)] \tag{6}$$

with $y_1(t) = \mathbf{w}_1(t)^{\mathrm{T}}\mathbf{x}(t)$. This is iterated over the training set $\mathbf{x}(1),\mathbf{x}(2),\ldots$. The parameter $\gamma(t)$ is the learning rate controlling the speed of convergence of $\mathbf{w}_1(t)$ to $\mathbf{e}_1$ as $t \to \infty$.

Likewise, taking the gradient of y_j^2 with respect to the weight vector $\mathbf{w}_j$ and using the constraints $\mathbf{w}_i^{\mathrm{T}}\mathbf{w}_j = \delta_{ij}$, we end up with the learning rule

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \gamma(t)y_j(t)\left[\mathbf{x}(t) - y_j(t)\mathbf{w}_j(t) - 2\sum_{i<j} y_i(t)\mathbf{w}_i(t)\right]. \tag{7}$$

On the right-hand side there is a term $y_j(t)\mathbf{x}(t)$, which is a so-called Hebbian term, product of the output y_j of the jth neuron and the input $\mathbf{x}$ to it. The other terms are

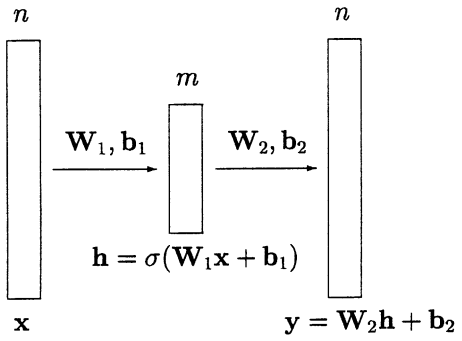

Fig. 4. The 3-layer MLP in autoassociative mode.

implicit orthonormality constraints. The case $j = 1$ gives the one-unit learning rule (6) of the basic PCA neuron. The convergence of the vectors $\mathbf{w}_1, \ldots, \mathbf{w}_m$ to the eigenvectors $\mathbf{e}_1, \ldots, \mathbf{e}_m$ was established in [36]. A modification called the generalized Hebbian algorithm (GHA) was later presented by Sanger [45], who also applied it to image coding, texture segmentation, and the development of receptive fields.

Other related on-line algorithms have been introduced in [14,44,12,53]. Some of them, like the APEX algorithm by Diamantaras and Kung [12], are based on a feedback neural network. Also minor components defined by the eigenvectors corresponding to the smallest eigenvalues can be computed by similar algorithms [38].

Another possibility for PCA computation in neural networks is the multi-layer perceptron network, which learns using the back-propagation algorithm (see [19]) in *unsupervised autoassociative mode*. The network is depicted in Fig. 4.

The input and output layers have n units and the hidden layer has $m < n$ units. The outputs of the hidden layer are given by

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \tag{8}$$

where $\mathbf{W}_1$ is the input-to-hidden layer weight matrix, $\mathbf{b}_1$ is the corresponding bias vector, and σ is the activation function, to be applied elementwise. The output $\mathbf{y}$ of the network is an affine linear function of the hidden layer outputs:

$$\mathbf{y} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2 \tag{9}$$

with obvious notation.

In autoassociative mode, the same vectors $\mathbf{x}$ are used both as inputs and as desired outputs in back-propagation learning. If σ is *linear*, then the hidden layer outputs will become the principal components of $\mathbf{x}$ [3]. For the linear network, back-propagation learning is especially feasible because it can be shown that the "energy" function has no local minima.

This network with nonlinear hidden layer was suggested for data compression by [10], and it was shown to be closely connected to the theoretical PCA by [7]. It is not equivalent to PCA, however, as shown by [27], unless the hidden layer is linear. A

much more powerful network is obtained if more hidden layers are added. For instance, a 5-layer autoassociative MLP is able to compute in principle *any* smooth nonlinear mapping between the inputs and the central hidden layer, and another mapping between the central hidden layer and the outputs. This is due to the two extra nonlinear hidden layers; see e.g. [37].

3.3. Factor analysis

In FA [18], a generative latent variable model is assumed for $\mathbf{x}$

$$\mathbf{x} = \mathbf{A}\mathbf{y} + \mathbf{n}. \tag{10}$$

FA was originally developed in social sciences and psychology. In these disciplines, the researchers want to find relevant and meaningful factors that explain observed results [18,30,54]. The interpretation in model (10) is that the elements of $\mathbf{y}$ are the unobservable factors.

The elements a_{ij} of the unknown matrix $\mathbf{A}$ are called *factor loadings*. The elements of the unknown additive term $\mathbf{n}$ are called specific factors. The elements of $\mathbf{y}$ (the factors) are uncorrelated, zero mean and gaussian, and their variances are absorbed into the matrix $\mathbf{A}$ so that we may assume

$$E\{\mathbf{y}\mathbf{y}^{\mathrm{T}}\} = \mathbf{I}. \tag{11}$$

The elements of vector $\mathbf{n}$ are zero mean, uncorrelated with each other and also with the factors y_i; denote $\mathbf{Q} = E\{\mathbf{n}\mathbf{n}^{\mathrm{T}}\}$. It is a diagonal matrix. We may write the covariance matrix of the observations from (10) as

$$E\{\mathbf{x}\mathbf{x}^{\mathrm{T}}\} = \mathbf{C}_{\mathbf{x}} = \mathbf{A}\mathbf{A}^{\mathrm{T}} + \mathbf{Q}. \tag{12}$$

In practice, we have a good estimate of $\mathbf{C}_{\mathbf{x}}$ available, given by the sample covariance matrix. The main problem is then to solve the matrix $\mathbf{A}$ of factor loadings and the diagonal covariance matrix $\mathbf{Q}$ such that they will explain the observed covariances from (12). There is no closed-form analytic solution for $\mathbf{A}$ and $\mathbf{Q}$. Assuming $\mathbf{Q}$ is known or can be estimated, we can solve $\mathbf{A}$ from $\mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{C}_{\mathbf{x}} - \mathbf{Q}$.

This solution is not unique, however: any matrix $\mathbf{A}' = \mathbf{A}\mathbf{T}$ where $\mathbf{T}$ is an orthogonal matrix ($\mathbf{T}\mathbf{T}^{\mathrm{T}} = \mathbf{I}$) will also be a solution. Then the factors will change to $\mathbf{y}' = \mathbf{T}^{\mathrm{T}}\mathbf{y}$. For $\mathbf{A}'$ and $\mathbf{y}'$, the FA model (10) holds, and the elements of $\mathbf{y}'$ are still uncorrelated. The reason is that the property of uncorrelatedness is invariant to orthogonal transformations (rotations). Note that because the factors are uncorrelated and gaussian, they are also independent.

3.4. Independent component analysis

In ICA [1,5,8,9,25,26,28,29,39] the same model (10) is assumed, but now the assumption on y_i is much stronger: we require that they are *statistically independent* and *nongaussian*. Interestingly, then the ambiguity in Factor Analysis disappears and the solution, if we can find one, is (almost) unique.

In the simplest form of ICA, the additive noise $\mathbf{n}$ is not included and the standard notation for the independent components or *sources* is s_i; thus the ICA model for observation vectors $\mathbf{x}$ is

$$\mathbf{x} = \mathbf{A}\mathbf{s}. \tag{13}$$

It is again assumed that both $\mathbf{x}$ and $\mathbf{s}$ are zero mean. The observations x_i are now linear combinations or mixtures of the sources s_j. The matrix $\mathbf{A}$ is called in ICA the *mixing matrix*.

We may further assume that the dimensions of $\mathbf{x}$ and $\mathbf{s}$ are the same. If originally $\dim \mathbf{x} < \dim \mathbf{s}$, or there are more sources than observed variables, then the problem becomes quite difficult—see [25]. If, on the other hand, $m = \dim \mathbf{x} > \dim \mathbf{s} = n$, then model (13) implies that there is redundancy in $\mathbf{x}$ which is revealed and can be removed by performing PCA on $\mathbf{x}$. This is done as follows.

We can write the $m \times m$ covariance matrix of $\mathbf{x}$ as

$$\mathbf{C}_\mathbf{x} = \mathbf{A}E\{\mathbf{s}\mathbf{s}^\mathrm{T}\}\mathbf{A}^\mathrm{T} = \mathbf{A}\mathbf{A}^\mathrm{T}. \tag{14}$$

We have used the knowledge that matrix $E\{\mathbf{s}\mathbf{s}^\mathrm{T}\}$ is diagonal, due to the fact that the elements of $\mathbf{s}$ are zero mean and independent; if we further absorb their variances to matrix $\mathbf{A}$ and assume that $E\{s_i^2\} = 1$, then it holds $E\{\mathbf{s}\mathbf{s}^\mathrm{T}\} = \mathbf{I}$. Now, matrix $\mathbf{A}$ is an $m \times n$ matrix and so matrix $\mathbf{C}_\mathbf{x} = \mathbf{A}\mathbf{A}^\mathrm{T}$ is an $m \times m$ matrix with rank n. It will have only n nonzero eigenvalues and the corresponding eigenvectors span a signal subspace. By projecting $\mathbf{x}$ onto this signal subspace, a reduced model is obtained in which the dimensions of $\mathbf{x}$ and $\mathbf{s}$ are both the same.

In fact, another step called *whitening* is very useful as a preprocessing stage in ICA, and it can be combined into the dimensionality reduction. Let us denote the diagonal matrix of the nonzero eigenvalues of $\mathbf{C}_\mathbf{x}$ by $\mathbf{D}$; as noted above, it will be an $n \times n$ matrix. Let us again denote the orthonormal eigenvectors of $\mathbf{C}_\mathbf{x}$ by $\mathbf{e}_1, \ldots, \mathbf{e}_m$ and the orthogonal matrix that has the n first ones as columns by $\mathbf{E}$. Thus $\mathbf{E}$ is $m \times n$. Make now a linear transformation for the m-dimensional observation vectors $\mathbf{x}$:

$$\mathbf{x}' = \mathbf{D}^{-1/2}\mathbf{E}^\mathrm{T}\mathbf{x}. \tag{15}$$

For the covariance matrix of the transformed n-dimensional vector $\mathbf{x}'$ it holds:

$$\begin{aligned} E\{\mathbf{x}'\mathbf{x}'^\mathrm{T}\} &= \mathbf{D}^{-1/2}\mathbf{E}^\mathrm{T}\mathbf{C}_\mathbf{x}\mathbf{E}\mathbf{D}^{-1/2} \\ &= \mathbf{D}^{-1/2}\mathbf{E}^\mathrm{T}\mathbf{E}\mathbf{D}\mathbf{D}^{-1/2} = \mathbf{I}. \end{aligned}$$

A vector whose covariance matrix is the unit matrix is called white. Whitening can be always performed because we only need the covariance matrix of observation vectors $\mathbf{x}$, which can be readily estimated from a sample. Alternatively, whitening can be done by neural network learning rules, that are simplifications of the on-line PCA learning rules [25]. Let us assume in the following that whitening has always been performed in the model, and denote simply by $\mathbf{x}$ the whitened observation vector whose dimension is the same as that of the source vector $\mathbf{s}$.

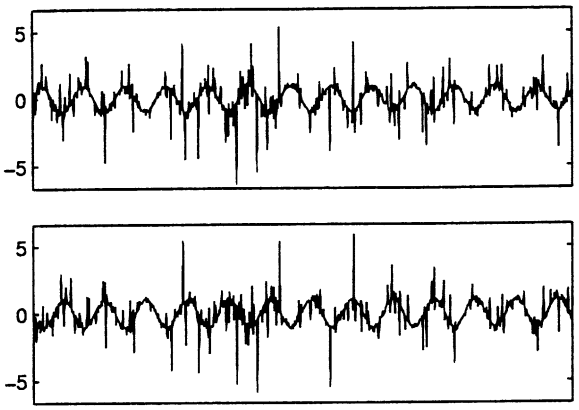

Fig. 5. Mixed signals.

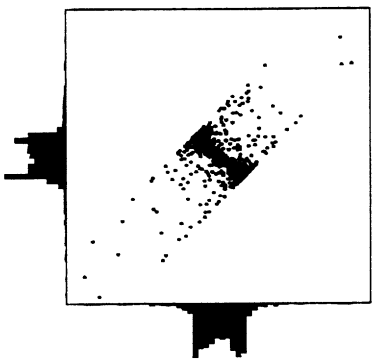

Fig. 6. Histogram of the two amplitudes of the mixed signals.

Whitening has another desirable side-effect, which can be seen by noting from Eq. (14) that now $\mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{I}$. But this means that matrix $\mathbf{A}$ is an orthogonal matrix, for which $\mathbf{A}^{-1} = \mathbf{A}^{\mathrm{T}}$. So, if we knew matrix $\mathbf{A}$, we could directly solve the unknown source vector $\mathbf{s}$ from the model by

$$\mathbf{s} = \mathbf{A}^{\mathrm{T}}\mathbf{x}.$$

It is an interesting finding that very few assumptions suffice for solving the mixing matrix and, hence, the sources. All we need is the assumption that the sources s_i are statistically independent and nongaussian. Consider the following simple example: we have two signals, shown in Fig. 5, that are linear combinations or mixtures of two underlying independent nongaussian source signals. This example is related to model (13) in such a way that the elements x_1, x_2 of the random vector $\mathbf{x}$ in (13) are the amplitudes of the two signals in Fig. 5. The signals provide a sample $\mathbf{x}(1), \ldots, \mathbf{x}(T)$ from this random vector. The joint histogram of the sample vectors is plotted in Fig. 6;

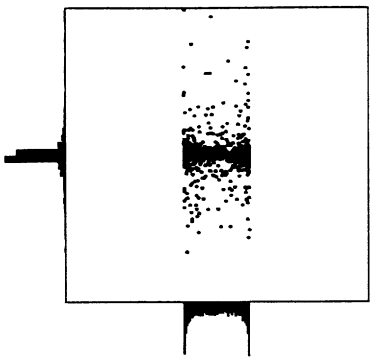

Fig. 7. Histogram of the two amplitudes of the separated signals.

each point in the scatter plot corresponds to one time point in Fig. 5. The vector $\mathbf{x}$ is now white in the sense that x_1 and x_2 are zero mean, uncorrelated, and have unit variance. This may not be apparent from the histogram but can be verified by estimating the covariance matrix of all the points.

The example suggests a method that in fact is highly useful and forms the basis of some practical ICA algorithms. Consider a line passing through the origin at the center of the data cloud in Fig. 6. Denote a unit vector defining the direction of the line by $\mathbf{w}$. Then the projection of a data point $\mathbf{x}$ on the line is given by $y = \mathbf{w}^T\mathbf{x}$. This can be considered as a random variable whose density is approximated by the histogram of the projections of all the data points in the cloud on this line. No matter what is the orientation of the line, it always holds that y has zero mean and unit variance. The unit variance is due to $E\{y^2\} = E\{(\mathbf{w}^T\mathbf{x})^2\} = \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w} = \mathbf{w}^T\mathbf{w} = 1$ where we have used the facts that $\mathbf{x}$ is white and $\mathbf{w}$ has unit norm.

However, it is easy to see from Fig. 6 that the density of y will certainly vary as the orientation of the line varies, meaning that all the moments of y cannot stay constant. In fact, any other moment than the first and second ones is not constant. What is most important is that any such moment, say, $E\{y^3\}$ or $E\{y^4\}$ or in fact $E\{G(y)\}$, with $G(y)$ a nonlinear and nonquadratic function, will attain a number of maxima and minima when the orientation of the line goes full circle, and some of these extrema coincide with orientations in which the 2-dimensional density factorizes into the product of its marginal densities—meaning independence.

In Fig. 7, the coordinate system has been rotated so that the fourth moment $E\{y^4\}$ is maximal in the vertical direction and minimal in the horizontal direction. We have found two new variables $y_1 = \mathbf{w}_1^T\mathbf{x}$ and $y_2 = \mathbf{w}_2^T\mathbf{x}$, with $\mathbf{w}_1, \mathbf{w}_2$ orthonormal, that satisfy

$$p(y_1, y_2) = p(y_1)p(y_2)$$

with $p(.)$ the appropriate probability densities. The variables are thus independent and it holds

$$\mathbf{y} = \mathbf{W}\mathbf{x},$$

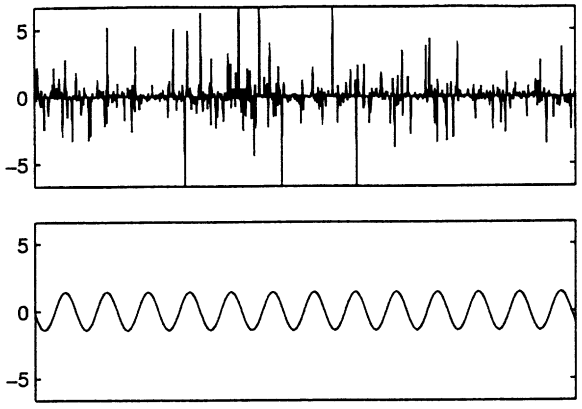

Fig. 8. Separated signals.

where $\mathbf{W}=(\mathbf{w}_1\mathbf{w}_2)^{\mathrm{T}}$. We have solved the inverse of model (13) and obviously found the mixing matrix: $\mathbf{A}=\mathbf{W}^{\mathrm{T}}$.

Fig. 8 shows y_1, y_2 again arranged in their correct time order. It is seen that they form two signals, one a random nongaussian noise and the other one a deterministic sinusoid. These were in fact the original signals that were used to make the artificial mixtures in Fig. 5. In the context of separating time series or signals, the ICA technique is an example of *blind signal separation*.

The above illustrative example can be formalized to an efficient mathematical algorithm. What we need is a numerical method to maximize, say, the fourth moment $E\{y^4\}$ in terms of a unit norm weight vector $\mathbf{w}$. A possibility is gradient ascent: the gradient of $E\{y^4\}$ with respect to $\mathbf{w}$ is $4E\{y^3\mathbf{x}\}=4E\{(\mathbf{w}^{\mathrm{T}}\mathbf{x})^3\mathbf{x}\}$. However, gradient methods are notoriously slow. A better idea is a fast algorithm with higher-order convergence speed. Such a method is provided by the FAstICA algorithm. For finding one independent component (one weight vector $\mathbf{w}$), the algorithm is as follows:

1. Choose the initial value randomly for the weight vector $\mathbf{w}$.
2. Repeat steps 3, 4 until the algorithm has converged.
3. Normalize $\mathbf{w}$ to unit norm.
4. Update $\mathbf{w}$ by

$$\mathbf{w}\leftarrow E\{(\mathbf{w}^{\mathrm{T}}\mathbf{x})^3\mathbf{x}\}-3\mathbf{w}. \tag{16}$$

This algorithm was introduced in [26] and further extended and analyzed in [24]; for a detailed review, see [25]. The FastICA algorithm is available in public-domain software [13] from the author's web pages. The algorithm can be run either in a deflation mode, in which the orthogonal weight vectors (columns of the mixing matrix $\mathbf{A}$) can be found one at a time, or in a parallel mode, in which all the independent components and the whole matrix $\mathbf{A}$ are solved in one iteration.

An analysis of the local maxima and minima of a general cost function $E\{G(y)\}=E\{G(\mathbf{w}^{\mathrm{T}}\mathbf{x})\}$ over the unit sphere $\|\mathbf{w}\|=1$ was made by the author in [41]. The result is

Theorem. *Under the linear mixing model* $\mathbf{x} = \mathbf{A}\mathbf{s}$, *with whitened* $\mathbf{x}$ (*hence*: *orthogonal* $\mathbf{A}$), *the local maxima* (*resp. minima*) *of* $E\{G(\mathbf{w}^{\mathrm{T}}\mathbf{x})\}$ *under the constraint* $\|\mathbf{w}\| = 1$ *include those columns* $\mathbf{a}_i$ *of the mixing matrix* $\mathbf{A}$ *such that the corresponding sources* s_i *satisfy*

$$E\{s_i g(s_i) - g'(s_i)\} > 0 \quad (resp. < 0), \tag{17}$$

where $g(.)$ *is the derivative of* $G(.)$.

The Theorem essentially says that all the columns of the mixing matrix will be among the local minima or maxima of $E\{G(\mathbf{w}^{\mathrm{T}}\mathbf{x})\}$, but there may be also other extrema. Condition (17) states that some columns (and the corresponding sources) are found by minimizing, others by maximizing. For the case $G(y) = y^4$, (17) becomes

$$E\{s_i^4 - 3\} > 0$$

(note that the sources have unit variances). The term on the left-hand side is the kurtosis of s_i. Thus, the positively kurtotic sources are found at the local maxima of $E\{(\mathbf{w}^{\mathrm{T}}\mathbf{x})^4\}$ and vice versa. For other cost functions $G(y)$, condition (17) always splits the sources in two groups, too.

In [25], the above method of fourth-order moment maximization is shown to be an example of a powerful criterion of finding *maximally nongaussian orthogonal directions* through the multidimensional density $p(\mathbf{x})$. Cost functions like maximum likelihood or minimal mutual information are shown to be intimately related to this basic criterion. Other algorithms to solving the basic linear ICA model have been reported, e.g. by [1,5,8,9,28], as reviewed in [25].

When the additive noise cannot be assumed to be zero in the ICA model, we have the noisy ICA model, also termed independent factor analysis [2]. This is due to the fact that it is otherwise similar to the factor analysis model (10), with the difference that the factors y_i are not uncorrelated (thus independent) gaussians, but rather independent nongaussians. Some solution methods are reviewed in [25]. Another extension is nonlinear ICA, discussed in Section 5.3.

4. The self-organizing map

One of the best-known neural networks in the unsupervised category is the SOM introduced by Kohonen [31]. It belongs to the class of *vector coding* algorithms. In vector coding, the problem is to place a fixed number of vectors, called *codewords*, into the input space which is usually a high-dimensional vector space. The input data are given as a training set $\mathbf{x}(1), \ldots, \mathbf{x}(T)$. For example, the inputs can be gray-scale windows from a digital image, measurements from a machine or an industrial process, financial data describing a company or a customer, or pieces of English text represented by word histograms. The dimension n of the data vectors is determined by the problem and can be very large. In the WEBSOM system [32] for organizing collections of text documents, the dimensionality of the data in the largest applications is about $n = 50,000$ and the size of the training sample is about $T = 7,000,000$.

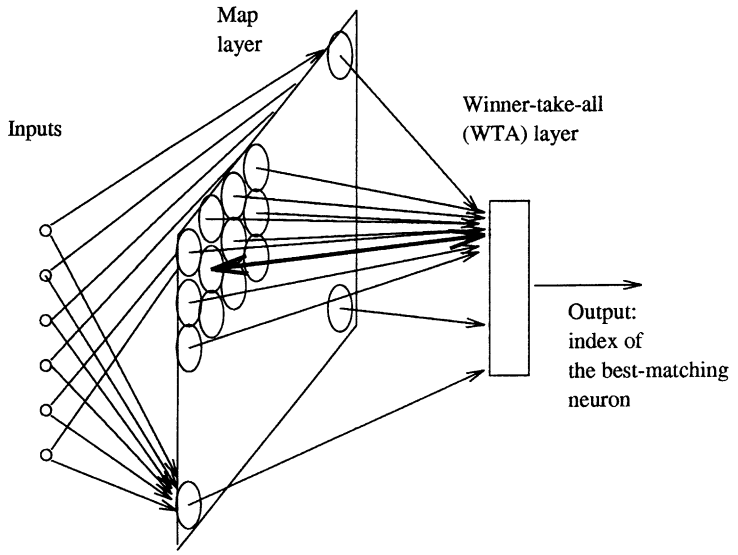

Fig. 9. The SOM network. Each neuron in the map layer receives the same inputs. The best matching neuron (BMU) can be found by a winner-take-all (WTA) layer which outputs its index. In learning, the BMU and its neighbors receive a learning signal from the WTA (only the signal to the BMU is shown by the thick arrow), telling them to update their weights.

A well-known method for vector coding is the Linde-Buzo-Gray (LBG) algorithm, which is very similar to the *k-means* clustering algorithm [46]. It is possible to interpret these as unsupervised neural learning rules. Assume a set of units (neurons) which are numbered by index $i = 1, \ldots, k$, and assume that each unit i has a weight vector $\mathbf{w}_i$ that has the same dimension as the input vectors $\mathbf{x}$ that we wish to cluster. In both the LBG algorithm and k-means clustering, the goal is to place the weight vectors (codewords) into the input space in such a way that the average squared distance from each $\mathbf{x}$ to its closest codeword is minimized. The learning algorithm is based on *competitive learning*, in which the codewords compete for the vectors and the winner is allowed to adapt. This is a very natural approach to neural learning. For instance, building on his early work on competitive learning, Grossberg introduced the adaptive resonance theory (ART) [16]. The ART network can be used for pattern recognition and clustering.

In the SOM introduced by Kohonen [31], there is an extra feature compared to mere clustering: the neurons (nodes) are arranged to a 1-, 2- or multidimensional *lattice*, such that each neuron has a set of *neighbors*; see Fig. 9. The goal of SOM learning is not only to find the most representative code vectors for the input training set in the sense of minimum distance, but at the same time to form a *topological mapping* from the input space to the grid of neurons. This idea originally stems from the modelling of the topographic maps on the sensory cortical areas of the brain. A related early work in neural modelling is [34].

For any data point $\mathbf{x}$ in the input space, one or several of the codewords are closest to it. Assume that $\mathbf{w}_i$ is the closest among all codewords:

$$\|\mathbf{x} - \mathbf{w}_i\| = \min \|\mathbf{x} - \mathbf{w}_j\|, \quad j = 1, \ldots, k. \tag{18}$$

The unit i having the weight vector $\mathbf{w}_i$ is then called the *best-matching unit* (BMU) for vector $\mathbf{x}$. Note that for fixed $\mathbf{x}$, Eq. (18) defines the index $i = i(\mathbf{x})$ of the BMU, and for fixed i, Eq. (18) defines the set of points $\mathbf{x}$ that are mapped to that index. By the above relation, the input vectors $\mathbf{x}$ are mapped to the discrete set of indices i.

By a topological mapping the following property is meant: if a given point $\mathbf{x}$ is mapped to unit i, then all points in neighborhoods of $\mathbf{x}$ are mapped either to i itself or to one of the units in the neighborhood of i in the lattice. Because no topological maps between two spaces of different dimensions can exist in the strict mathematical sense, a two-dimensional neural layer can only follow locally two dimensions of the multidimensional input space. Usually the input space has a much higher dimension, but the data cloud $\mathbf{x}(1), \ldots, \mathbf{x}(T)$ used in training may be roughly concentrated on a lower-dimensional manifold that the map is able to follow at least approximately [31].

The SOM network is shown in Fig. 5. The role of the output 'winner-take-all' layer is to compare the outputs from the map layer (equivalently, the distances $\|\mathbf{x} - \mathbf{w}_i\|$) and give out the index of the BMU. The well-known Kohonen algorithm for self-organization of the code vectors is as follows [31]:

1. Choose initial values for the weight vectors $\mathbf{w}_i$.
2. Repeat steps 3, 4 until the algorithm has converged.
3. Draw a sample vector $\mathbf{x}$ from the training set and find the best matching unit $i = i(\mathbf{x})$ according to Eq. (18).
4. Adjust the weight vectors of all units by

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + \gamma * h_r * (\mathbf{x} - \mathbf{w}_j) \tag{19}$$

where γ is a gain factor and h_r is a function of the distance $r = \|i - j\|$ of units i and j measured along the lattice.

There are several choices for the initial values and for the neighborhood; these, as well as the convergence and the mathematical properties of this algorithm have been considered by several authors, e.g. [31,42,40,51]. For SOM learning, topology preservation, and its relation to a cost function, see [52,11,20]. A more efficient learning rule for the SOM is the *batch algorithm*, covered, e.g. in [31]. The 2-dimensional map is also a powerful tool for data visualization: e.g., a color code can be used in which each unit has its own characteristic color. For a public domain software implementation of the SOM, with various graphical tools for map presentations as well as with preprocessing methods, see [48]. A database of well over three thousand applications of SOM is given by [47].

5. Generative models

The concept of a generative model is very general and potentially powerful. In fact, as discussed by Roweis and Ghahramani [43], a large number of central techniques like

FA, PCA, ICA, mixtures of Gaussians, vector quantization, and also dynamical models like Kalman filters or Hidden Markov Models, can be presented in a unified framework of unsupervised learning under a single basic generative model. In his Bayesian Ying–Yang model, Xu [55] likewise has presented a generic framework of unsupervised learning for the basic data models, both static and temporal.

In this review, we concentrate on generative models for the central neural network techniques reviewed in the previous sections: FA, PCA, ICA, and topographic maps. We already saw examples of generative models in the case of FA and ICA. Also PCA can be derived from a generative model in the technique called probabilistic PCA [49] or principal factor analysis [18].

A problem with such linear models is that they cannot represent well data that is not a linear mixture of some underlying gaussian or nongaussian variables. For data clouds that have an irregular or curved shape, these methods fail. A possible remedy is to use a piecewise linear model, or a mixture of local linear models. The probabilistic approach allows the extension of PCA to such a mixture model, which is covered first in Section 5.1. There is an approximative generative model for the SOM, too, called the generative topographic map (GTM). It will be reviewed in Section 5.2. Then, recent nonlinear generalizations of FA and ICA are reviewed in Section 5.3.

5.1. Mixtures of principal component analyzers

A fairly obvious general idea of combining the two major unsupervised learning paradigms—PCA and vector coding (VQ)—is to use mixtures of linear models, for example PCA's, in which the data cloud is first clustered or parcelled using VQ, and then a separate linear model is fitted to each of the clusters around the code vector. This notion has been formalized by several authors [21,42,43,49,55]. For instance, consider the FA model

$$\mathbf{x} = \mathbf{A}\mathbf{y} + \mathbf{n} \tag{20}$$

with both $\mathbf{y}$ and $\mathbf{n}$ zero mean and gaussian, with $E\{\mathbf{y}\mathbf{y}^{\mathrm{T}}\} = \mathbf{I}$ and $\mathbf{Q} = E\{\mathbf{n}\mathbf{n}^{\mathrm{T}}\}$. From this, the gaussian conditional density $p(\mathbf{x}|\mathbf{y})$ is directly obtained, and this immediately gives $p(\mathbf{x})$ because $p(\mathbf{y})$ is a gaussian of simple form. If there are several FA models instead, with different mean values, then the data obeys a mixture density

$$p(\mathbf{x}) = \sum_{i=1}^{m} \pi_i \, p(\mathbf{x}|i). \tag{21}$$

The log-likelihood can be derived directly. Tipping and Bishop [49] consider this under the constraint that in each model i, the noise co-variance is $\mathbf{Q}_i = \sigma_i^2\mathbf{I}$—they call this the probabilistic PCA model. They develop an iterative EM algorithm for optimizing all the model parameters (the weights π_i, the means, the factor loading matrices $\mathbf{A}_i$, and the noise variances σ_i^2). The obtained model can be used for *local* dimensionality reduction in cases where a single linear PCA model would fail completely.

This approach is closely related to the conventional technique of semiparametric density estimation, the *mixture of gaussians* (MoG) model. However, instead of using

full covariance matrices for the component gaussians, the local linear models constrain the covariances in a natural and easily adjustable way.

5.2. Generative topographic map

In the Generative topographic map (GTM) algorithm [6], the vectors $\mathbf{x}$ are expressed in terms of a number of latent variables, which are defined on a similar lattice or grid as the neurons in the SOM. Assume a neuron grid with dimension l (usually, this would be equal to 2, at most 3), and assume there are k nodes $\mathbf{y}_i$, $i = 1,\ldots,k$ on the grid. Assume a random latent variable $\mathbf{y}$, whose values are concentrated at these nodes. We can then express the density of $\mathbf{y}$ as

$$p(\mathbf{y}) = \frac{1}{k}\sum_{i=1}^{k}\delta(\mathbf{y}-\mathbf{y}_i)$$

with $\delta(.)$ the Dirac delta function. Let us now make a nonlinear mapping from the l-dimensional random variable $\mathbf{y}$ to the original n-dimensional vectors $\mathbf{x}$:

$$\mathbf{x} = \mathbf{f}(\mathbf{y},\mathbf{M}) + \mathbf{n}, \tag{22}$$

where $\mathbf{M}$ is an array of parameters of the nonlinear function $\mathbf{f}$, and $\mathbf{n}$ is additive noise. The form of the function $\mathbf{f}$ is assumed to be determined except for the unknown parameters. Model (22) is the generative latent variable model of the GTM method. It means that the data $\mathbf{x}$ are basically concentrated on an l-dimensional nonlinear manifold in the data space, except for the additive noise. The k vectors $\mathbf{w}_i = \mathbf{f}(\mathbf{y}_i,\mathbf{M})$ that are the images of the node points $\mathbf{y}_i$ are analogous to the weight vectors or codewords of the SOM. If $\mathbf{f}$ is smooth, a topographic ordering for the codewords is automatically guaranteed, because such an ordering is valid for the points $\mathbf{y}_i$. The GTM also has the advantage that it postulates a smooth manifold that naturally interpolates between the code vectors $\mathbf{w}_i$.

If we assume that the noise has a radially symmetrical gaussian density with variance β^{-1}, the density of $\mathbf{x}$, given $\mathbf{y}$, is

$$p(\mathbf{x}|\mathbf{y},\mathbf{M},\beta) \propto \exp\left\{-\frac{\beta}{2}\|\mathbf{x}-\mathbf{f}(\mathbf{y},\mathbf{M})\|^2\right\} \tag{23}$$

from which the unconditional density is obtained as

$$p(\mathbf{x}|\mathbf{M},\beta) = \int p(\mathbf{x}|\mathbf{y},\mathbf{M},\beta)\,p(\mathbf{y})\,\mathrm{d}\mathbf{y} = \frac{1}{k}\sum_{i=1}^{k} p(\mathbf{x}|\mathbf{y}_i,\mathbf{M},\beta).$$

This density is a mixture of gaussians, having a separate gaussian density around each of the code vectors $\mathbf{w}_i = \mathbf{f}(\mathbf{y}_i,\mathbf{M})$. From this, the likelihood function for the parameters $\mathbf{M}$, β follows immediately. The EM algorithm can now be used to numerically solve the parameters by maximum likelihood, due to the mixture of gaussians form of the density—for details, see [6]. The reference also discusses the similarities and differences between GTM and SOM.

5.3. Nonlinear factor and independent component analysis

When comparing the FA model (20) and the GTM model (22), certain similarities emerge: both have a number of latent variables, given by the vector $\mathbf{y}$, and additive gaussian noise $\mathbf{n}$. In FA, the mapping from $\mathbf{y}$ to the data $\mathbf{x}$ is linear, in GTM it is nonlinear. Another clear difference is that in FA, the factors are gaussian, while in GTM, the prior density $p(\mathbf{y})$ for the latent factors has a very special form.

Another possibility for this density in the nonlinear case, too, would be the gaussian density, which would be closer to the original flavor of FA. If we assume that the prior for $\mathbf{y}$ is gaussian with unit (or diagonal) covariance, making the elements y_i independent, model (22) may be called *nonlinear factor analysis.* A further extension would be $p(\mathbf{y})$ that is nongaussian but factorizable so that the y_i are independent; then the model becomes *nonlinear independent component analysis.*

Recently, Valpola [33] used an approximation for the nonlinear function $\mathbf{f}(\mathbf{y},\mathbf{M})$ in the model, that was based on a Multi layer Perceptron (MLP) network with one hidden layer. It is well known [23,15] that this function can approximate uniformly any continuous functions on compact input domains and it is therefore suitable for this task. Then the model becomes

$$\mathbf{x} = \mathbf{B}\phi(\mathbf{A}\mathbf{y} + \mathbf{a}) + \mathbf{b} + \mathbf{n}, \tag{24}$$

where $\mathbf{A}$, $\mathbf{a}$ are the weight matrix and offset vector of the hidden layer, ϕ is the sigmoidal nonlinearity, typically a tanh or $\sinh^{-1}$ function, and $\mathbf{B}$, $\mathbf{b}$ are the weight matrix and offset vector of the linear output layer. It is understood that ϕ is applied to its argument vector element by element. In practice, there is a training sample $\mathbf{x}(1),\ldots,\mathbf{x}(T)$, and we wish to solve from the model the corresponding source or factor vectors $\mathbf{y}(1),\ldots,\mathbf{y}(T)$.

The problem now is that, contrary to the usual supervised learning situations, the inputs to the MLP are not known and therefore back-propagation type of learning rules cannot be used for finding the unknown parameters. The idea in [33] is to use a purely Bayesian approach called *ensemble learning.* The cost function is the Kullback–Leibler divergence between the true posterior probability for the parameters, given the observations, and an approximation of that density. Denote the set of all the unknown parameters by $\mathbf{\Theta} = \{\mathbf{Y},\mathbf{M}\}$. There the vector $\mathbf{Y}$ contains all the unknown source vectors $\mathbf{y}(1),\ldots,\mathbf{y}(T)$, while $\mathbf{M}$ contains the weights of the MLP network that define the unknown nonlinear function $\mathbf{f}$, and also the parameters of the gaussian noise $\mathbf{n}$. In addition, because this is a Bayesian model, it includes hyperparameters defining the distributions of the weights. Denote the sample of observations by $\mathbf{X} = \mathbf{x}(1),\ldots,\mathbf{x}(T)$.

We can write for the posterior density of the parameters

$$p(\Theta|\mathbf{X}) = p(\mathbf{Y},\mathbf{M}|\mathbf{X}) = \frac{p(\mathbf{X}|\mathbf{Y},\mathbf{M})\,p(\mathbf{Y}|\mathbf{M})\,p(\mathbf{M})}{p(\mathbf{X})}. \tag{25}$$

The first term $p(\mathbf{X}|\mathbf{Y},\mathbf{M})$ is obtained from the data model; it is simply a product of gaussians with means $\mathbf{B}\phi(\mathbf{A}\mathbf{y}(t) + \mathbf{a}) + \mathbf{b}$. Likewise, the terms $p(\mathbf{Y}|\mathbf{M})$ and $p(\mathbf{M})$ are obtained as products of gaussians, when we assume mutually independent gaussian

priors for all the parameters. The term $p(\mathbf{X})$ does not contain any unknown parameters and can be omitted.

This density is now approximated by another density $q(\mathbf{\Theta})$—the ensemble—that has a simple form [33]: it is a gaussian with diagonal covariance. Then the KL divergence

$$C_{KL} = \int \mathrm{d}\mathbf{\Theta}\, q(\mathbf{\Theta}) \log \frac{q(\mathbf{\Theta})}{p(\mathbf{\Theta}|\mathbf{X})} \tag{26}$$

also obtains a relatively simple form, splitting into the expectations of many simple terms. It can be minimized by a suitable numerical method.

In [50], several applications with real data are shown. The model is also extended to a dynamical model, similar to an extended Kalman filter but with unknown parameters, and very promising results are obtained in case studies.

6. Conclusions

The two main paradigms of unsupervised neural learning have been reviewed: the extensions to the principal component analysis (PCA) technique, and the neural vector coding and topological mapping technique. The first class of methods form a continuous linear or nonlinear transformation of the original input vectors to feature vectors of lower dimensionality, and are especially useful in feature extraction. The reduced representation given by the feature vectors would typically be input to another network, e.g. a classifier. Both the PCA networks and the autoassociative multi-layer perceptron networks were reviewed.

The second class of methods are able to map highly nonlinear input data manifolds onto low-dimensional neural lattices, preserving optimally the mutual topological relations of input vectors. Thus these methods, notably the self-organizing map are suitable for data clustering and visualization. The applications range from industrial quality control to financial data mining. Also generative latent variable versions for these basic models and their combinations were reviewed.

This paper was a review of the essential principles and theory underlying unsupervised learning, with some central references cited. It is not possible here to give even a rudimentary list of applications of these techniques. There are available good text-books that cover some of the major approaches [22,31,12,25].

References

[1] S.-I. Amari, A. Cichocki, H. Yang, A new learning algorithm for blind source separation, in: D.S. Touretzky, M.C. Moser, M.E. Hasselmo (Eds.), Advances in Neural Information Processing Systems, Vol. 8, MIT Press, Cambridge, MA, 1996, pp. 757–763.
[2] H. Attias, Independent factor analysis, Neural Comput. 11 (4) (1999) 803–851.
[3] P. Baldi, K. Hornik, Learning in linear neural networks: a survey, IEEE Trans. Neural Networks 6 (4) (1995) 837–858.
[4] H. Barlow, Unsupervised learning, Neural Comput. 1 (1989) 295–311.
[5] A. Bell, T. Sejnowski, An information-maximization approach to blind separation and blind deconvolution, Neural Comput. 7 (1995) 1129–1159.

[6] C. Bishop, M. Svensen, C. Williams, GTM: the generative topographic mapping, Neural Comput. 10 (1998) 215–234.
[7] H. Bourlard, Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biol. Cybernet. 59 (1988) 291–294.
[8] J.-F. Cardoso, Blind signal separation: statistical principles, Proc. IEEE 9 (10) (1998) 2009–2025.
[9] A. Cichocki, R. Unbehauen, Robust neural networks with on-line learning for blind identification and blind separation of sources, IEEE Trans. Circuits and Systems 43 (11) (1996) 894–906.
[10] G. Cottrell, P. Munro, D. Zipser, Learning internal representations from gray-scale images: an example of extensional programming, Proc. 9th Ann. Conf. of the Cognitive Science Society, Seattle, WA, 1987, pp. 462–473.
[11] R. Der, M. Herrmann, Second-order learning in self-organizing maps, in: E. Oja, S. Kaski (Eds.), Kohonen Maps, Elsevier, Amsterdam, 1999, pp. 293–302.
[12] K. Diamantaras, S. Kung, Principal Component Neural Networks: Theory and Applications, Wiley, New York, 1996.
[13] The FastICA package, Available from www.cis.hut.fi/projects/ica/.
[14] P. Foldiak, Adaptive network for optimal linear feature extraction, Proc. Internat. J. Conf. on Neural Networks, Washington, DC, 1989, pp. 401–406.
[15] K. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (1989) 183–192.
[16] S. Grossberg, Direct perception or adaptive resonance? Behav. Brain Sci. 3 (1980) 385.
[17] S. Grossberg, Neural Networks and Natural Intelligence, MIT Press, Cambridge, MA, 1988.
[18] H.H. Harman, Modern Factor Analysis, University of Chicago Press, Chicago, 1967.
[19] S. Haykin, Neural Networks–Comprehensive Foundation, MacMillan College Publ. Co., New York, 1998.
[20] T. Heskes, Energy functions for self-organizing maps, in: E. Oja, S. Kaski (Eds.), Kohonen Maps, Elsevier, Amsterdam, 1999, pp. 303–316.
[21] G. Hinton, M. Revow, P. Dayan, Recognizing handwritten digits using mixtures of linear models, in: G. Tesauro, D. Touretzky, T. Leen (Eds.), Advances in Neural Information Processing Systems, Vol. 6, Kauffman, San Mateo, 1995, pp. 1015–1022.
[22] G. Hinton, T.J. Sejnowski, Unsupervised Learning—Foundations of Neural Computation, MIT Press, Cambridge, MA, 1999.
[23] M. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–368.
[24] A. Hyvärinen, Fast and robust fixed-point algorithms for independent component analysis, IEEE Trans. Neural Networks 10 (3) (1999) 626–634.
[25] A. Hyvärinen, J. Karhunen, E. Oja, Independent Component Analysis, Wiley, New York, (2001).
[26] A. Hyvärinen, E. Oja, A fast fixed-point algorithm for independent component analysis, Neural Comput. 9 (7) (1997) 1483–1492.
[27] N. Japkowitz, S. Hanson, A. Gluck, Nonlinear autoassociation is not equivalent to PCA, Neural Comput. 12 (3) (2000) 531–545.
[28] C. Jutten, J. Herault, Blind separation of sources, part I: an adaptive algorithm based on neuromimetic architecture, Signal Process. 24 (1991) 1–10.
[29] J. Karhunen, E. Oja, L. Wang, R. Vigario, J. Joutsensalo, A class of neural networks for independent component analysis, IEEE Trans. Neural Networks 8 (3) (1997) 486–504.
[30] M. Kendall, A. Stuart, The Advanced Theory of Statistics, Vols. 1–3, MacMillan, New York, 1976–1979.
[31] T. Kohonen, Self-Organizing Maps, Springer, Berlin, 1995. (3rd Edition, 2001)
[32] T. Kohonen, S. Kaski, K. Lagus, J. Salojärvi, V. Paatero, A. Saarela, Self organization of massive document collection, IEEE Trans. Neural Networks 11 (3) (2000) 574–585.
[33] H. Lappalainen, A. Honkela, Bayesian nonlinear independent component analysis by multi-layer perceptrons, in: M. Girolami (Ed.), Advances in Independent Component Analysis, Springer, Berlin, 2000, pp. 93–121.
[34] C. von der Malsburg, Self-organization of orientation sensitive cells in the striate cortex, Kybernetik 14 (1973) 85–100.

[35] E. Oja, A simplified neuron model as a principal components analyzer, J. Math. Biol. 15 (1982) 267–273.
[36] E. Oja, Subspace Methods of Pattern Recognition, RSP and Wiley, Letchworth, 1983.
[37] E. Oja, Data compression, feature extraction, and autoassociation in feedforward neural networks, Proc. ICANN-91, Espoo, Finland, June 24–28, 1991, pp. 737–745.
[38] E. Oja, Principal components, minor components, and linear neural networks, Neural Networks 5 (1992) 927–935.
[39] E. Oja, The nonlinear PCA learning rule in independent component analysis, Neurocomputing 17 (1) (1997) 25–46.
[40] E. Oja, S. Kaski (Eds.), Kohonen Maps, Elsevier, Amsterdam, 1999.
[41] E. Oja, L. Wang, Neural fitting: robustness by anti-Hebbian learning, Neurocomputing 12 (1976) 155–170.
[42] H. Ritter, T. Martinetz, K. Schulten, Neural Computation and Self-Organizing Maps: An Introduction, Addison-Wesley, Reading, MA, 1992.
[43] S. Roweis, Z. Ghahramani, A unifying review of linear Gaussian models, Neural Comput. 11 (2) (1999) 305–346.
[44] J. Rubner, P. Tavan, A self-organizing network for principal component analysis, Europhys. Lett. 10 (7) (1989) 693–698.
[45] T. Sanger, Optimal unsupervised learning in a single-layered linear feedforward network, Neural Networks 2 (1989) 459–473.
[46] R. Schalkoff, Pattern Recognition—Statistical, Structural, and Neural Approaches, Wiley, New York, 1992.
[47] Bibliography of SOM papers: a reference list of over 4000 studies on the self-organizing map, Available at www.cis.hut.fi/research/som-bibl/.
[48] The SOM Toolbox for Matlab, Available at www.cis.hut.fi/projects/somtoolbox/.
[49] M.E. Tipping, C.M. Bishop, Mixtures of probabilistic principal component analyzers, Neural Comput. 11 (2) (1999) 443–482.
[50] H. Valpola, Bayesian ensemble learning for nonlinear factor analysis, Acta Polytech. Scand. Math. 108 (2000); D.Sc. Thesis, Helsinki University of Technology.
[51] M. VanHulle, Faithful Representations and Topographic Maps, Wiley, New York, 2000.
[52] T. Villmann, Topology preservation in self-organizing maps, in: E. Oja, S. Kaski (Eds.), Kohonen Maps, Elsevier, Amsterdam, 1999, pp. 267–292.
[53] L. Wang, J. Karhunen, A unified neural bigradient algorithm for robust PCA and MCA, Internat. J. Neural Systems 7 (1) (1996) 53–67.
[54] A. Webb, Statistical Pattern Recognition, Arnold, Paris, 1999.
[55] L. Xu, Temporal BYY learning for state space approach, hidden Markov model, and blind source separation, IEEE Trans. Signal Process. 48 (2000) 2132–2144.