International Journal of Neural Systems, Vol. 1, No. 1 (1989) 61–68
© World Scientific Publishing Company

NEURAL NETWORKS, PRINCIPAL COMPONENTS, AND SUBSPACES

Erkki Oja
Lappeenranta University of Technology
Department of Information Technology
Box 20, 53851 Lappeenranta, Finland

Received 6 April 1989

A single neuron model with Hebbian-type learning for the connection weights, and with nonlinear internal feedback, has been shown to extract the statistical principal components of its stationary input pattern sequence. A generalization of this model to a layer of neuron units is given, called the Subspace Network, which yields a multi-dimensional, principal component subspace. This can be used as an associative memory for the input vectors or as a module in nonsupervised learning of data clusters in the input space. It is also able to realize a powerful pattern classifier based on projections on class subspaces. Some classification results for natural textures are given.

1. Introduction

Training strategies of artificial neural networks for signal processing, pattern recognition and associative memories belong to one of two basic categories: supervised and unsupervised. The unsupervised learning neural networks try to model the space of input vectors. An essential component is then a "similarity detector" unit. The present author introduced one such model, the principal component extractor,[1] which can be used as a building block in self-organizing nets.[2,3,4] It is based on Hebbian learning of the connection weights with an extra nonlinear feedback term which imposes an automatic constraint on the individual weights. Thus, if the inputs are suitably bounded, the output from the unit stays within a given interval even if there is no saturating nonlinearity at the outputs.

An obvious generalization of the principal component unit would be a network of several units with simultaneous Hebbian learning that implements a multi-dimensional principal component subspace. The mathematical algorithm for it was given by the author in Ref. 5 and a network implementation, the Subspace Network, was proposed in the unpublished report.[6] It is reviewed here. It is a one-layer net of either linear or nonlinear interconnected units all sharing the same inputs. The learning rule is a generalization of that in Ref. 1. The weight vectors are autonomously adapted to the input data in the following way: first, each weight vector is normalized to a fixed length; second, the weight vectors of different units become orthogonal; and third, the vectors tend to be tuned to the statistically most important feature dimensions of the inputs. Again, there is neither any explicit normalization nor orthogonalization for the weight vectors but these effects are an automatic result of the specific nonlinear feedback term in the learning rule.

Such a network could function as an associative memory for its input data. In a pattern recognition application, several network modules can be used for supervised learning of preclassified samples or for nonsupervised learning of data clusters in the input space. This corresponds to the subspace method of pattern recognition,[5,7] recently advocated in Ref. 8, which was originally motivated by associative memory neural networks. However, a direct neural network implementation for the learning subspace classifier has not been given before.

It seems that recently there has been some renewed interest in the study of essentially linear networks which are also in the supervised learning category. They may serve as a basis for understanding qualitatively the behavior of the more efficient nonlinear networks whose analysis is otherwise difficult. It turns out that a multilayer feedforward net with back-propagation learning then reduces to a discriminant analysis classifier[9] and in the autoassociative case the outputs from the hidden layer are the principal components of the input vector.[10,11] Especially the latter result is very similar to the one obtained from the Subspace Network although the starting point for the training strategy is quite different.

From a mathematical point of view, the learning algorithms of the networks reviewed in the following have been presented earlier by the author in the context of pattern recognition and signal processing.[5,12] Recently Sanger[13] has also proposed network implementations for some of the modifications and discussed their relationship with the linear back-propagation network.

2. The Basic Unit

Following Kohonen,[2] the basic model neuron is defined as follows (see Fig. 1).

The unit receives n input signals denoted by ξ_i, with $i = 1, \ldots, n$, which affect the unit through connection weights. On the jth unit these are denoted μ_{ji}, $i = 1, \ldots, n$. We assume here that both the inputs and the weights can attain both positive and negative continuous values. The inputs and weights influence the output of the unit, η_j, by their linear cumulative effect

$$\nu_i = \sum_{i=1}^{n} \mu_{ji}\xi_i \, . \tag{1}$$

This is the integrated effect of the inputs on the unit. The output of the unit is controlled by a dynamical equation

$$\frac{d\eta_j}{dt} = \nu_j - \gamma(\eta_j) \, . \tag{2}$$

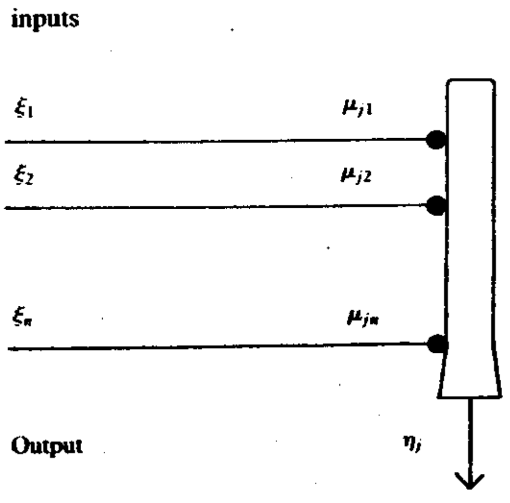

Fig. 1. The basic neuron.

There η_j is the actual output of the unit and γ is a nonlinear leakage effect. The inverse function of $\gamma(.)$, denoted by $\sigma(.)$, typically has a sigmoidal form. Thus the short-time stationary output signal from Eq. 2, $\eta_j = \sigma(\nu_j)$ will be a nonlinear function of the linearly weighted input signals, as is usually the case in artificial neuron models.

The time constant of Eq. 2 is very fast as compared to the time constant of learning in which the weights μ_{ji} change. The learning is assumed to take place according to a Hebbian type conjunction of the inputs ξ_i and the integrated effect of the inputs, ν_j, with an extra "forgetting" term:

$$\frac{d\mu_{ji}}{dt} = \alpha\nu_j\xi_i - f(\nu_j, \xi_i, \mu_{ji}) \, . \tag{3}$$

In the simplest case, $f(.)$ might be a constant. However, quite interesting functions emerge from the assumption that it depends on the various factors in the unit, especially on ν_j. This helps in stabilizing the weights to a constrained range of values.

3. The Principal Component Analyzer

As a simple but nontrivial special case of Eq. 3, assume that $f(.)$ has the form proposed by the author in Ref. 1:

$$f(\nu_j, \xi_i, \mu_{ji}) = \beta\nu_j^2\mu_{ji} \tag{4}$$

with β constant, which results in the learning equation

$$\frac{d\mu_{ji}}{dt} = \nu_j(\alpha\xi_i - \beta\nu_j\mu_{ji}) \, . \tag{5}$$

Each weight μ_{ji} tends to grow according to its input ξ_i, but the growth is controlled by an internal feedback in the unit. The feedback gets stronger with a stronger ν_j.

Changing to a vector-matrix notation in which the column vectors m_j, x are

$$m_j = (\mu_{j1}, \ldots, \mu_{jn})^T, \; x = (\xi_1, \ldots, \xi_n)^T, \tag{6}$$

it holds that $\nu(t) = m(t)^T x(t)$ (index j is dropped for the time being) and Eq. 5 becomes

$$\frac{dm(t)}{dt} = \alpha x(t)x(t)^T m(t) - \beta m(t)^T x(t)x(t)^T m(t)m(t) \, . \tag{7}$$

This is completely equivalent to the original learning equation (5).

Sometimes it can be assumed that the input vector stays stationary through the learning period, i.e., the values are picked from an n-dimensional distribution characterizing the variations within some signal class. Then a statistical analysis becomes possible. Taking averages in Eq. 7 (for details of the averaging process, see Ref. 12) and redefining the constants α and β to be the same for simplicity (this is not a crucial assumption) leads to

$$\frac{dm}{dt} = Cm - (m^T Cm)m \,. \tag{8}$$

The $n \times n$ symmetric and positive definite matrix C is the autocorrelation matrix of the inputs:

$$C = E[x(t)x(t)^T], \; [C]_{pq} = E[\xi_p \xi_q]. \tag{9}$$

The learning equation (8) has been analyzed by the author in Refs. 1 and 12 and the result is:

As t tends to infinity, The Euclidean norm of vector $m(t)$ tends to one and $m(t)$ tends to c_1 (or $-c_1$), the eigenvector of C corresponding to the largest eigenvalue, if $m(0)$ is not orthogonal to c_1. This is the coefficient vector of the largest principal component in the inputs; in other words, c_1 defines that one-dimensional subspace on which the squared magnitude of the projection of input x is maximal on the average.

The latter property means that among all unit vectors u, the vector c_1 maximizes the "energy" functional $E(u^T x)^2$. But $(m(t)^T x)^2 = \nu^2$ which is therefore maximized on the average. Thus also the output η of the unit will be large if the input is similar to the inputs which occurred during learning. The unit acts as a *principal component analyzer*, a *similarity detector*, and a *matched filter for its inputs*.

Note that the negative internal feedback stabilizes the weights in the sense that their squared sum over the unit stays constant *independently of the strengths and values of the inputs* even though there is no explicit normalization.

In fact several such units, each tuned to a different signal class, could already be used as a classifier network, since with a high probability the strongest response for an input vector to be classified would be obtained from the unit sensitive to the class of the input vector. This type of units have been used by Kohonen[2] in his self-organizing network and by Linsker[4] in his perceptual feature-detection network.

4. The Subspace Network

4.1. *The learning equation*

Subspaces and projection operations have an important role in the theories of distributed data processing and associative memories.[3] The same formalism also applies to pattern recognition.[5,8] Many network models have been introduced earlier for the implementation of the linear optimal projection mappings for signal patterns.[3,14]

Consider the net of k closely interacting neural units of Fig. 2. For each unit, the integrated effect of the inputs, ν_j is again obtained from Eq. 1. The synaptic weights vary again according to the Hebbian law with internal feedback, expressed in Eq. 3, but now the feedback is more complicated. It is not restricted to individual units, but the ν_j terms of all the units give rise to a net effect reducing the strength of the input signal:

$$\frac{d\mu_{ji}}{dt} = \alpha \nu_j(\xi_i - \zeta_i) \tag{11}$$

with the internal feedback effect ζ_i given by

$$\zeta_i = \sum_{h=1}^{k} \mu_{hi} \nu_h \,. \tag{12}$$

The feedback effect ζ_i which reduces the strength of the ith input ξ_i is obtained as the weighted sum of all factors ν_h, h ranging over the units in the net. In Hebbian terms, Eq. 11 means that an input strengthens a weight on a unit if both the input and the net effect of all the inputs and weights of that unit are high, but this effect is reduced if the weights of the other units connecting with that input line are already strong.

Another way to visualize Eqs. 1, 11 and 12 as a network topology would be a rectangular crossbar

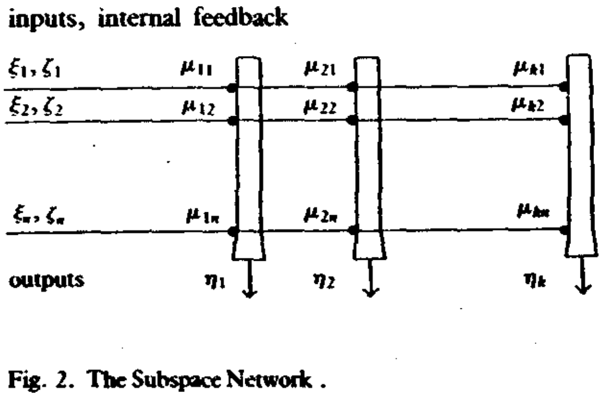

Fig. 2. The Subspace Network .

switching net with the signals ζ_i as actual *outputs* and the signals ν_j mediating in the input-output function: first the inputs ξ_i are integrated (vertically in Fig. 2) into signals ν_j according to Eq. 1, then each output line computes its own weighted sum (horizontally) according to Eq. 12. This would be a purely linear net.

Equation 11 has now quite interesting mathematical consequences. Reverting again to vector-matrix notation, define the n-vector z as

$$z = (\zeta_1, \ldots, \zeta_n)^T, \tag{13}$$

the k-vector v as

$$v = (\nu_1, \ldots, \nu_k)^T, \tag{14}$$

and the $n \times k$-matrix M as

$$M = [\mu_{ji}] = (m_1, \ldots, m_k). \tag{15}$$

The columns of M are the weight vectors of the units. Then Eq. 1 becomes

$$v = M^T\mathbf{x}, \tag{16}$$

Eq. 12 becomes

$$z = Mv, \tag{17}$$

and the learning equation (11) becomes

$$\frac{dM}{dt} = \alpha(\mathbf{x} - z)v^T. \tag{18}$$

Note that if z were considered the actual output vector from the net, then the total linear input-output function would be

$$z = MM^T\mathbf{x}. \tag{19}$$

Developing Eq. 18 further yields

$$\frac{dM}{dt} = \alpha(\mathbf{x} - MM^T\mathbf{x})\mathbf{x}^TM$$

$$= \alpha[\mathbf{x}\mathbf{x}^TM - M(M^T\mathbf{x}\mathbf{x}^TM)]. \tag{20}$$

Again, if the input data arrive as samples from some stationary pattern class distribution with autocorrelation matrix given by Eq. 9, then Eq. 20 can be averaged to yield

$$\frac{dM}{dt} = \alpha[CM - M(M^TCM)]. \tag{21}$$

There has been some analysis of this and related learning rules by the present author[12] and by Karhunen.[15] Due to the third-degree matrix polynomial in Eq. 21, the equation is difficult to solve and a rigorous mathematical analysis does not yet exist, although the corresponding equation for $M M^T$ can be solved in closed form under certain initial conditions. The properties obtained from the learning equation (21) have been studied by computer simulations and they indicate that the behavior is analogical to that of the one-dimensional case of Eq. 8. The results can be summarized as follows:

Matrix $M(t)$ tends to have orthonormal columns and as t grows, the columns of $M(t)$ span the same subspace as vector $c_1, \ldots, c_k$ which are the eigenvectors of C corresponding to the k largest eigenvalues. These are the coefficient vectors of the k largest principal components in the inputs; in other words, $c_1, \ldots, c_k$ define that unique k-dimensional subspace on which the squared magnitude of the projection of input $\mathbf{x}$ is maximal on the average.

The last property means that matrix MM^T tends to a projection operator on the k-principal component subspace. If $z = MM^T\mathbf{x}$ is considered as the actual output from the network, then it gives the corresponding projection of input vector $\mathbf{x}$. Also, internally, the squared sum of the terms ν_j represents the projection magnitude since

$$\sum_{j=1}^{k} \nu_j^2 = \|v\|^2 = \mathbf{x}^TMM^T\mathbf{x} = \mathbf{x}^TMM^TMM^T\mathbf{x}$$

$$= \|MM^T\mathbf{x}\|^2 = \|z\|^2. \tag{22}$$

If the functional in Eq. 22, or some monotonic function of it, can be computed from the outputs of the units, then the network acts as a powerful similarity detector for inputs belonging to the same statistical distribution as the inputs during learning.

Note that again there is no explicit normalization or orthogonalization in algorithm (11). It follows entirely from the special type of feedback within the net. Due to this term, the connection weights stay bounded, the weight vectors of the different units within the net become orthonormal, and the weight matrix becomes a k-principal component detector.

The learning rule (11) has been tested for some data sets. In Ref. 15 a simple set of artificial stationary input data was used. The differential equation was discretized. The input vectors x_k were five-dimensional and the eigenvalues of their correlation matrix C were $\lambda_1 = 0.853$, $\lambda_2 = 0.301$, $\lambda_3 = 0.163$, $\lambda_4 = 0.053$, and $\lambda_5 = 0.030$. The number of units in the cluster, k, was chosen as 3. Equation 19 (equivalent with (11)) was discretized to the form

$$M_k = M_{k-1} + \alpha[x_k x_k^T M_{k-1} - M_{k-1}(M_{k-1}^T x_k x_k^T M_{k-1})] \qquad (23)$$

with α being a small constant. The vectors x_k were picked randomly from their distribution. Initially, the weight vectors m_1, m_2, and m_3 were randomly chosen vectors of unit length; it turns out that they had to be at least roughly orthogonal at the start.

0	0.5758	0.8420	0.6593
40	0.8540	0.8378	0.7510
80	0.9299	0.9124	0.9333
120	0.9984	0.9759	0.9785
160	0.9827	0.9962	0.9702
200	0.9761	0.9839	0.9880

Table I. Comparison of the weight vectors with the correct eigenvector subspace. First column: iteration number. Second, third, and fourth columns: projections of m_1, m_2, and m_3, respectively, on the correct eigenvector subspace. A value of one denotes complete fit.

Knowing the exact distribution of the inputs, it is possible also to compute the eigenvectors of the correlation matrix C. Then it is easy to test whether the weight vectors, in the course of time, become orthonormal and span the same subspace as the three dominant eigenvectors of C. Table I shows the projections of m_1, m_2, and m_3 on this theoretically correct subspace. After 200 input vectors have been used in algorithm (22), the projections are already close to 1. As for the orthonormality, Table II shows the matrix $M^T_{200} M_{200}$. If the columns of M are orthonormal, then this matrix is the unit matrix. The matrix of Table II is a good approximation to the unit matrix. The small differences in Tables I and II as compared to the theoretically exact results are due to random fluctuations in the elements of M_k (the connection weights) caused by the varying input data. With a suitably decreasing parameter α the errors would tend to zero. Other simulations with high-dimensional real input vectors (texture data) showed exactly the same behavior.

$$\begin{pmatrix} 1.015 & -0.0061 & 0.0016 \\ -0.0061 & 1.020 & -0.0066 \\ 0.0016 & -0.0066 & 1.011 \end{pmatrix}$$

Table II. Matrix $M^T_{200} M_{200}$. Elements are inner products of weight vectors m_1, m_2, and m_3.

4.2 *The Subspace Network as a classifier*

In pattern recognition, the primary goal is to build an explicit or implicit *model* for the pattern classes such that the classification error for new data is minimized, or equivalently, the generalization ability of the classifier is maximized. A geometrical viewpoint is often useful. Given the representation for the classes and the distance function or other classification rule, the class regions are completely defined in the high-dimensional pattern space.

In the subspace methods of pattern recognition,[5,8] the primary model for a class is a subspace, and the classification criterion for a pattern x is its orthogonal distance from the class subspaces: x is classified to the class which gives the shortest such distance. The subspace defines a class as the general *linear combination* of some spanning basis vectors. This view, although certainly not generally valid for any type of data, seems to be particularly useful for patterns like *spectra* and *histograms*. Still, it must be emphasized that the class boundaries of the method are generally nonlinear.[5]

Consider, for example, the acoustic *power spectrum* emitted by some object. The elements of vector x are then the energies in the different spectral bands. Assume that the spectra consist of a set of characteristic modes at fixed frequencies but with intensities that are excited randomly. Then an arbitrary realization of the spectrum is obviously a linear combination of a finite set of basis vectors. Classification of objects is very similar to fitting the spectrum x to the different class models in order to find out which model could best explain the observed x. This is exactly equivalent to subspace projections. As indicated above, the Subspace Network is able to derive such models from its input data with suitable learning strategies.

Imagine now that in a supervised classification situation with K classes, a network consists of K subnetworks or *modules* similar to the one described above. Within each module, the numbers of units $k^{(1)}, \ldots, k^{(K)}$ may be different but, for simplicity, assume in the following that each module has exactly k units. Each subnetwork receives the same inputs. The *training* of the classifier is as follows:

1. Input a training vector to all modules. At the same time, input a special *training input* to all modules; it is one for the module corresponding to the correct class for the input vector (call it module *a*) and zero otherwise.

2. Form the output of each module as a monotonically increasing function of the term $\sum_{j=1}^{k} \nu_j^2$ (which, according to Eq. 21, gives the projection magnitude). Find the module that gave the largest output (call it module *b*).

3. If module *a* and module *b* are the same (correct classification for the training input vector), then modify the weights of that module according to Eq. 11 with a positive value for α, increasing the similarity of the training vector with the subspace represented by that module.

4. If module *a* and module *b* are not the same (incorrect classification for the training input vector), then there are at least two feasible strategies:

4a. Modify only module *a* with positive α.

4b. Modify module *a* with positive α and module *b* with negative α, increasing the similarity of the training vector with the subspace of module *a* and decreasing it with that of module *b*.

The training is iterated several times over all the input vectors. A good choice for the initial values of the weight vectors for each module is a set of random vectors of unit length, which should be roughly orthonormal within each module but otherwise independently chosen within different modules.

In fact strategy 4a corresponds to the so-called Clafic algorithm,[16] strategy 4b to the learning subspace methods (LSM,[7] ALSM[5]) within the methodology of subspace classifiers.

5. Classification Results

The classification results given in the following have mostly been obtained by using the practical short-cut algorithm ALSM, explained in detail in Ref. 5, instead of simulating the Subspace Network as such. However, in some cases at least the network and the computational algorithm are equivalent.

Some applications of the subspace method have been speech recognition[2,7] and color discrimination.[17] Recently, an extensive series of tests have been conducted on the classification and segmentation of textures using the subspace formalism; for a review, see Ref. 18 and the references therein. One test, explained in more detail in Ref. 19, is briefly explained here.

Texture features are known to contain significant discriminatory information for image segmentation in a variety of applications like terrain classification from remote sensing images, industrial robot vision, and biomedical image analysis. Some of the commonly used statistical texture features are based on Fourier spectra, gray level cooccurrence and difference histograms, run length matrices, spatial domain filter masks or random field models like autoregressive or Markov models.[20] The popularity of feature sets based on second-order statistics originates from studies of the human visual system.[21]

As test material, 10 texture images from the standard source, the Brodatz album,[22] were used (numbers D16, D33, D34, D49, D57, D68, D77, D84, D93, and D103). Each image was digitized to $256 \times 256 \times 8$ bits. The textures are shown in Fig. 3.

5.1. *Feature extraction.*

The images were compressed by histogram equalization[23] into 4 grey levels and $N \times N$ windows were chosen at random locations. In these experiments, 23×23 texture windows were used. Each

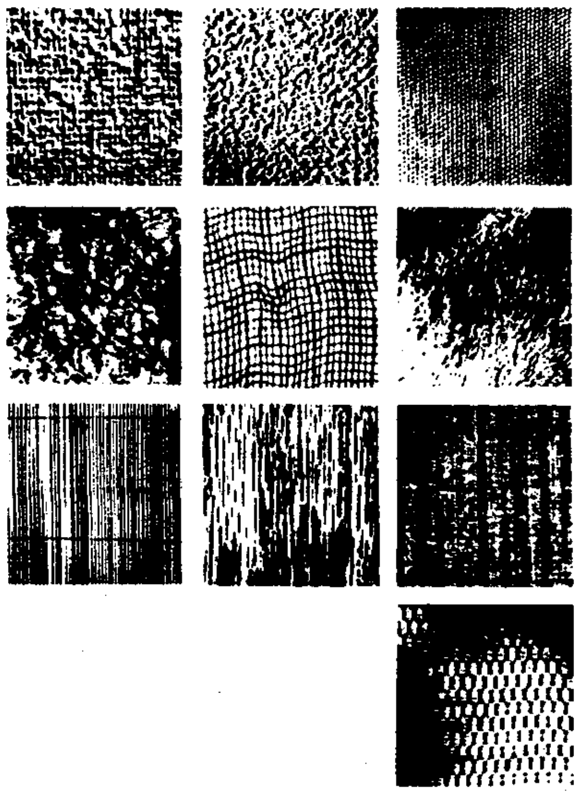
Fig. 3. The ten Brodatz textures used as test material.

such texture window was represented by a feature vector for subspace classification using either the Subspace Network or an approximating algorithm. The feature extraction was as follows:

1. A number of symmetric *cooccurrence matrices* with varying displacements[20] were computed for each texture window. The cooccurrence matrix $A(i, j; dk, dl)$ for horizontal displacement dk and vertical displacement dl is defined as follows: the ijth element of the matrix is equal to the number of pixel pairs in the sampling window such that the displacements between the two pixels are dk and dl, and one pixel has grey level i, the other has grey level j.

Since the number of grey levels was 4, the matrix size is 4×4. For these textures, it turned out that four good displacements are $(\pm 1,0)$, $(2,\pm 2)$, $(0,\pm 3)$, and $(\pm 3,2)$.

2. Each of the four cooccurrence matrices for a given texture was then stacked row by row into a 16 component vector, and these four vectors were concatenated to form a 64 component feature vector for the 23×23 texture window.

Such 64 component vectors, representing second-order statistics of the textures in different orientations and displacements, are believed to capture effectively a large amount of the discriminatory information contained in the texture window. They are fairly insensitive to the texture window size and location and, despite their relatively large dimensionality, are still conveniently classified with the subspace method. Such vectors were the input for the classifier.

5.2. *Classification*

Two independent sets of 80 texture windows from each of the 10 textures were taken, and one of them (containing 10×80 windows in all) was used as the design set, the other as the test set for classifier error estimation. In the short-cut algorithm, the class correlation matrices $C^{(i)}$ were directly computed from the training data together with their eigenvalues and eigenvectors. The eigenvectors $c^{(i)}{}_j$ corresponding to the k largest eigenvalues are now the representation for the ith class subspace and they were used to compute the discriminant functions $\sum_{j=1}^{k} (x^T c^{(i)}{}_j)^2$. The Subspace Network approximates the same eigenvector subspace after learning according to strategy 4a, as indicated in Sec. 4.

In testing the classifier, the classification accuracy for the training data was 96 % and for the test data 90 %. The difference reflects the fact that the sample size, restricted by the sizes of the texture windows, was too small for a direct error estimation procedure. For completely reliable error estimation, a bootstrapping technique in which the existing sample vectors are resampled should be used.

The results can be further improved by the Averaged Learning Subspace Method (ALSM) given in Ref. 5. After one iteration step the classification accuracy for the design set used in the iteration was 99 %. The corresponding accuracy for the test set was 93 %. The results did not improve with further iterations. A similar improvement is expected from the Subspace Network if the training strategy 4b of Sec. 4.2. is used, since it corresponds to the Learning Subspace Methods.

A comparison to other classifiers is presented in Ref. 19. It shows that the subspace method is a good classifier for textures.

References

1. E. Oja, "A simplified neuron model as a principal component analyzer", *J. Math. Biol.* **15** (1982) 267 – 273.
2. T. Kohonen, "The neural phonetic typewriter", *Computer* 21 (1988) 11 – 22.
3. T. Kohonen, *Self-Organization and Associative memory* (2nd Ed.). (Springer, Berlin, 1988).
4. R. Linsker, "Self-organization in a perceptual network", *Computer* 21 (1988) 105 – 117.
5. E. Oja, *Subspace Methods of Pattern Recognition*. (RSP and J. Wiley, 1983).
6. E. Oja, "Learning neural networks for pattern recognition", *Proc. ATR Workshop on Neural Networks and Parallel Distributed Processing*, Osaka, July 7 – 8, 1988, 33 – 34 (Abstract only).
7. T. Kohonen, H. Riittinen, M. Jalanko, E. Reuhkala, and S. Haltsonen, "A thousand-word recognition system based on the learning subspace method and redundant hash addressing", *Proc. 5 ICPR*, Miami Beach, 1980, 159 – 165.
8. E. Oja and T. Kohonen, "The subspace learning algorithm as a formalism for pattern recognition and neural networks", *Proc. IEEE 1988 ICNN*, San Diego, July 24 – 27, 277 – 284.
9. P. Gallinari, S. Thiria and F. Fogelman-Soulie, "Multilayer perceptrons and data analysis", *Proc. IEEE 1988 ICNN*, San Diego, July 24 – 27, 391 – 399.
10. P. Baldi and K. Hornik, "Neural networks and principal components analysis: learning from examples without local minima", *Neural Networks* **2** (1989) 53 – 58.
11. H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition", *Biol. Cyb.* **59** (1988) 291 – 294.
12. E. Oja and J. Karhunen, "On stochastic approximation of eigenvectors and eigenvalues of the expectation of a random matrix", *J.M.A.A.* **106** (1985) 69 – 84.

13. T. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network", to be published.
14. E. Oja, "Studies of the convergence properties of adaptive orthogonalizing filters", Dr. Tech. Thesis, Helsinki University of Technology, 1977.
15. J. Karhunen, "On the recursive estimation of the eigenvectors of correlation type matrices" (in Finnish), Tech.Lic. Thesis, Helsinki University of Technology, 1982.
16. S. Watanabe, *Knowing and Guessing* (Wiley, New York, 1969).
17. J. Parkkinen, E. Oja, and T. Jääskeläinen, "Color analysis by learning subspaces and optical processing", *Proc. IEEE 1988 ICNN*, San Diego, July 24–27, 421–428.
18. E. Oja, J. Parkkinen, K. Selkäinaho and T. Kärki, "Regularity measurement, classification, and segmentation of textures", *Proc. COST13 workshop* "From the pixels to the features", Bonas, France, Aug. 22–27, 1988.
19. E. Oja and T. Kärki, "Properties of the ALSM subspace classifier for texture analysis", *Proc. 5 SCIA*, Stockholm, 1987, 419–426.
20. R. M. Haralick, "Statistical and structural approaches to texture", *Proc. IEEE* 67 (1979) 786–804.
21. T. Julesz, "Visual pattern discrimination", *IRE Trans. Inf. Theory* IT-8 (1962) 84–92.
22. P. Brodatz, *Textures — A Photographic Album for Artists and Designers* (Reinhold, New York, 1968).
23. A. Rosenfeld and A. Kak, *Digital Picture Processing*. (Academic Press, New York, 1976).