Pergamon

Neural Networks, Vol. 7, No. 1, pp. 113–127, 1994

0893-6080/94 $6.00 + .00

CONTRIBUTED ARTICLE

Representation and Separation of Signals Using Nonlinear PCA Type Learning

JUHA KARHUNEN AND JYRKI JOUTSENSALO

Helsinki University of Technology

(*Received* 4 *February* 1993; *accepted* 9 *June* 1993)

Abstract—*A class of nonlinear PCA (principal component analysis) type learning algorithms is derived by minimizing a general statistical signal representation error. Another related algorithm is derived from a nonlinear feature extraction criterion. Several known algorithms emerge as special cases of these optimization approaches that provide useful information on the properties of the algorithms. By taking into account higher-order statistics, nonlinear algorithms are often able to separate component signals from their mixture. This is not possible with linear principal component subspace estimation algorithms. A suitably chosen nonlinearity makes the results more robust against various types of noise. Estimation of noisy sinusoids is used as a demonstration example.*

Keywords—Neural network, Unsupervised learning, Nonlinearity, Principal components, Signal representation, Blind separation, Oja's rule, Sinusoids.

1. INTRODUCTION

In unsupervised learning, the neural network tries to self-organize so that it detects some useful features, regularities, correlations, or signals from the input data with little or no knowledge on the desired results. Hebbian learning is a basic unsupervised technique especially in feedforward networks (Hertz, Krogh, & Palmer, 1991). This is because it often produces optimal responses to the input data and is neurobiologically justified (Linsker, 1988; Oja, 1982). In particular, Oja has shown in a seminal paper (Oja, 1982) that simple Hebbian learning applied to a single linear neuron extracts the feature describing best in the mean-square error sense the input data, namely the first principal component.

In Hebbian learning, the learning term is proportional to the product of the input and output of a neuron. If applied directly, this leads to two problems: 1) learning is unstable, that is, the weight vectors of the neurons grow infinitely large or tend to zero; 2) weight vectors of different neurons become similar, producing similar outputs. The first problem can be solved by adding some kind of stabilizing term. The latter problem can be managed either by competition of neurons or by imposing some constraint that makes the weight vectors different.

Oja's single neuron learning can be generalized to networks having several neurons in the output layer. The weight vectors of neurons converge then usually to the so-called PCA subspace of input vectors, which is defined by the principal eigenvectors of the correlation matrix of the input data. An example of this and the starting point of our study is the symmetric algorithm (Hertz et al., 1991; Oja, 1989)

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{I} - \mathbf{W}_k\mathbf{W}_k^T]\mathbf{x}_k\mathbf{x}_k^T\mathbf{W}_k. \qquad (1)$$

In eqn (1) the $L \times M$-matrix $\mathbf{W}_k = [\mathbf{w}_k(1), \ldots, \mathbf{w}_k(M)]$, $L \geq M$, has the weight vectors of the M neurons after k iterations as its columns. The output $\mathbf{x}_k^T\mathbf{w}_k(i)$ of ith neuron depends linearly on the corresponding L-dimensional neuron weight vector $\mathbf{w}_k(i)$ and kth input vector $\mathbf{x}_k$. A Hebbian type term $\mathbf{x}_k\mathbf{x}_k^T\mathbf{W}_k$, product $[\mathbf{x}_k^T\mathbf{w}_k(i)]\mathbf{x}_k$ of the output and the input of each neuron, is responsible for the learning. The gain parameter $\mu_k \geq 0$ controls the learning rate. The additive nonlinear constraint $\mathbf{W}_k\mathbf{W}_k^T\mathbf{x}_k\mathbf{x}_k^T\mathbf{W}_k$ roughly orthonormalizes the weight vectors: $\mathbf{W}_k^T\mathbf{W}_k \approx \mathbf{I}$.

The algorithm (1) can be shown to learn the M-dimensional PCA subspace of the input vectors (Baldi & Hornik, 1991; Hertz et al., 1991; Hrycej, 1992; Oja, 1992), even though a strict convergence proof is still lacking. If $\mathbf{W}_k$ has only one column, eqn (1) reduces to the basic Oja's single neuron algorithm. The algorithm (1) was first published in Oja (1989), though it

Requests for reprints should be sent to Dr. Juha Karhunen, Helsinki University of Technology, Laboratory of Computer and Information Science, Rakentajanaukio 2 C, SF-02150 Espoo, Finland.

had been studied several years earlier in context with Oja's single neuron learning rule (Karhunen, 1982; Addi, 1984) and independently by Williams (1985). The network computing eqn (1) can be realized in different ways (Oja, 1989); hardware implementation has been considered by Kotilainen, Saarinen, and Kaski (1992).

Many different more or less neural PCA subspace or principal components estimation algorithms have been proposed during the last years [for reviews and introduction, see Baldi & Hornik (1991); Cichocki & Unbehauen (1993); Hertz et al. (1991); Kung (1993); Oja (1992)]. Principal component analysis networks are useful in optimal feature extraction and data compression, and they have a number of possible applications in different areas. However, they have some limitations that make them less attractive from a neural network point of view.

1. PCA networks can realize only linear input–output mappings. Additional hidden layers with linear processing do not bring anything new. On the other hand, nonlinearity is one of the essential features of neural networks and a major reason for using them.
2. Computationally efficient standard numerical methods are available for solving the principal eigenvectors and -values needed in PCA. The relatively slowly converging gradient type neural algorithms are not always competitive with classical techniques especially in large problems, even though they were realized using hardware.
3. Principal components are based solely on covariances or correlations. These second-order statistics can describe completely Gaussian data and stationary, linear processing operations only. Therefore, higher-order statistical methods are currently becoming important in signal processing.
4. The outputs of PCA networks are mutually uncorrelated, but usually not independent. Even though the input data consist of a mixture of statistically independent subsignals, the uncorrelated outputs are generally some linear combinations of the subsignals only. In many cases the subsignals themselves are desired, but PCA networks cannot provide them directly.

For these reasons, it is meaningful to study various nonlinear generalizations of PCA learning algorithms and networks. If defined sensibly, such a learning algorithm could yield something *more* than just uncorrelated outputs that describe best, in the mean-square error sense, the input data. Some work has already been done in this direction. Carlson (1990) and Földiak (1990) have applied nonlinearity in context with anti-Hebbian learning. Sanger (1991) has added a nonlinear layer to a PCA network for approximating nonlinear functions. Softky and Kammen (1991) and Taylor and Coombes (1993) have used higher-order correlations in context with Oja's single neuron rule. White (1992) has derived an approximative nonlinear Hebbian learning rule by considering a constrained optimization problem. Shapiro and Prügel-Bennett (1992) have shown that if the activation function of the neuron is nonlinear enough, the neuron can learn to discriminate one pattern from the others; otherwise, it will learn a complex mixture of the input patterns. Sirat (1991) has used a simple hard limiting nonlinearity and shown that the results approximate PCA.

In an interesting paper by Oja, Ogawa, and Wangviwattana (1991), some nonlinear generalizations of the symmetric algorithm (1) have been proposed at the end of the paper, though their paper deals mostly with the single neuron case. The nonlinear variants have been obtained rather heuristically simply by replacing either one, two, or three of the products $\mathbf{x}_k^T\mathbf{W}_k$ or $\mathbf{W}_k^T\mathbf{x}_k$ appearing in eqn (1) by the nonlinearity $g(\mathbf{x}_k^T\mathbf{W}_k)$ or $g(\mathbf{W}_k^T\mathbf{x}_k)$. This leads to the following algorithms (in the original paper only the corresponding averaged differential equations are given):

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{x}_k\mathbf{x}_k^T\mathbf{W}_k - \mathbf{W}_k\mathbf{W}_k^T\mathbf{x}_k g(\mathbf{x}_k^T\mathbf{W}_k)] \quad (2)$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{I} - \mathbf{W}_k\mathbf{W}_k^T]\mathbf{x}_k g(\mathbf{x}_k^T\mathbf{W}_k) \quad (3)$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{x}_k g(\mathbf{x}_k^T\mathbf{W}_k) - \mathbf{W}_k g(\mathbf{W}_k^T\mathbf{x}_k) g(\mathbf{x}_x^T\mathbf{W}_k)]. \quad (4)$$

Here the function $g(t)$ is applied separately to each component of the argument vector. Thus, for example, $g(\mathbf{x}_k^T\mathbf{W}_k)$ is a row vector with elements $g[\mathbf{x}_k^T\mathbf{w}_k(i)]$, $i = 1, \ldots, M$. The same convention holds for other functions defined later. For stability reasons, $g(t)$ must be positive for positive argument values and negative for negative argument values. Usually $g(t)$ is a monotonic odd function, for example, $g(t) = \tanh(\alpha t)$, where α is a scalar parameter. In the algorithms (1)–(4), initial values of the weight vectors (columns of $\mathbf{W}_1$) may be chosen either randomly or by (ortho)normalizing first sample vectors, which yields sometimes faster convergence.

The structure of the neural network is shown in Figure 1. In the learning phase, the output of each neuron is connected to the inputs of all the other neurons. After learning, these feedback connections (shown by dashed lines) are not needed, and the network becomes purely feedforward. The same structure can be used for all the algorithms (1)–(4). In eqns (1) and (2), the output of the ith neuron is linear $\mathbf{x}^T\mathbf{w}(i)$ and in eqns (3) and (4) nonlinear $g[\mathbf{x}^T\mathbf{w}(i)]$.

A desirable property of eqns (1)–(4) is their symmetry in the sense that the form of each algorithm is the same for different neurons (columns of the matrix $\mathbf{W}_k$). Then no external hardwiring is required for realizing the corresponding neural circuit (Xu, 1991). Also, the weight vectors learn in parallel rather than sequentially, which is typically the case in algorithms containing some kind of hierarchy (Karhunen & Jout-

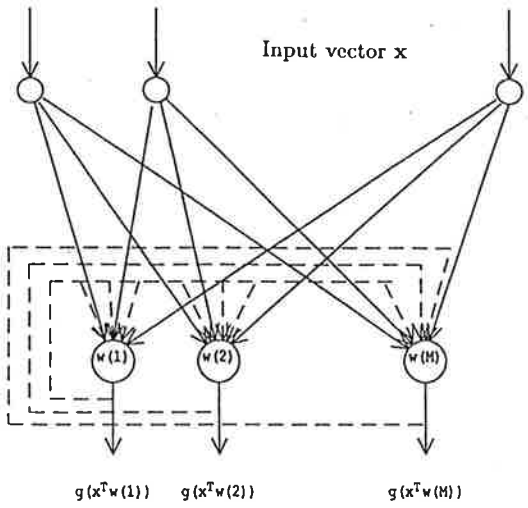

FIGURE 1. Architecture of the network. Feedback connections shown by dashed lines are needed only during the learning phase.

sensalo, 1991a). However, eqn (1) converges to some arbitrary orthonormal basis of the PCA subspace only. Generally, in algorithms having linear neuron input–output relationship, asymmetry (hierarchy) is required for learning the principal eigenvectors themselves (Baldi & Hornik, 1991).

An advantage of using nonlinearities is that the neurons seem to become more selective during the learning phase even though symmetry is preserved (Xu, 1991). Preliminary experimental results (Oja, Ogawa, & Wangviwattana 1992) show that the weight vectors given by eqn (2) converge towards the true nonnormalized principal eigenvectors of the correlation matrix of the input vectors if a sigmoidal nonlinearity is used. Note that the Hebbian learning part $\mathbf{x}_k\mathbf{x}_k^T\mathbf{W}_k$ in eqn (2) is still linear. Therefore, nothing more than a PCA solution is obtained in spite of the nonlinearity appearing in the constraint term.

The possible convergence points of the algorithms (2)–(4) are the asymptotic solutions of the corresponding averaged differential equations. These equations can be written easily (Oja et al., 1991), but it is usually impossible to determine their asymptotic solutions analytically even in the single neuron case. Preliminary experiments show that eqns (3) and (4), having a nonlinearity in the Hebbian learning term, do not generally converge to the PCA solution (Oja et al., 1991), even though in special cases the learning result may be much the same (Karhunen & Joutsensalo, 1992a).

Exact mathematical analysis of the algorithms (2)–(4) is rather difficult because of the nonlinearities. We have earlier studied their properties using a specific problem, sinusoidal frequency estimation (Karhunen & Joutsensalo, 1992a,b). Somewhat surprisingly, it turned out that the nonlinear versions of eqns (2)–(4) performed relatively well in white noise also. In this case, the theoretically optimal so-called MUSIC estimator used by us is given by the PCA subspace. In impulsive and colored noise, the nonlinear versions yielded, in many cases, better results than the linear algorithm in eqn (1), showing their potential usefulness.

In the next section, we first derive a new class of nonlinear PCA type learning algorithms by minimizing a general statistical signal representation error. The input signal vectors $\mathbf{x}$ are represented in a linear basis, but the coefficients of the expansion are generally nonlinear. The algorithms (4) and (1) are obtained as particular approximations of this algorithm class. In Subsection 2.3, eqns (3) and (1) are derived from a nonlinear feature extraction criterion. The optimization criteria yield a good insight into what the algorithms are actually doing and help to understand their properties. Various properties and interpretations of the nonlinear algorithms are discussed more closely in Section 3. In Section 4, we give experimental results and demonstrate that certain nonlinear PCA algorithms are useful in blind separation of component signals from their mixture. This problem has received some attention in signal processing research recently. In the last section, we present the conclusions of this study.

2. DERIVATION OF ALGORITHMS FROM NONLINEAR OPTIMIZATION CRITERIA

2.1. Minimization of Signal Representation Error

Consider the problem of approximating an L-vector $\mathbf{x}$ in terms of M basis vectors $\mathbf{w}(1), \ldots, \mathbf{w}(M)$, $M \leq L$. The coefficients $f_2[\mathbf{x}^T\mathbf{w}(i)]$ of the expansion depend generally nonlinearly on the inner product of $\mathbf{x}$ and the respective basis vector $\mathbf{w}(i)$. Then $\mathbf{x}$ can be expressed in the form

$$\mathbf{x} = \hat{\mathbf{x}} + \mathbf{e} = \sum_{i=1}^{M} f_2[\mathbf{x}^T\mathbf{w}(i)]\mathbf{w}(i) + \mathbf{e}, \qquad (5)$$

where $\mathbf{e}$ is error vector, and the summation defines the approximation $\hat{\mathbf{x}}$ of $\mathbf{x}$. The expansion can be written more compactly

$$\mathbf{x} = \mathbf{W}f_2(\mathbf{W}^T\mathbf{x}) + \mathbf{e}. \qquad (6)$$

We assume that $\mathbf{x}$ has zero mean and that $f_2(t)$ is a continuously differentiable, monotonically increasing odd function, though all these assumptions are not absolutely necessary. The choice $f_2(t) = t$ yields linear coefficients as a special case. The form of the expansion (5) is justified later in this paper.

Now one can define a general nonlinear statistical error criterion

$$J(\mathbf{W}) = \mathbf{1}^T E\{f_1(\mathbf{e})|\mathbf{W}\}$$
$$= \mathbf{1}^T E\{f_1[\mathbf{x} - \mathbf{W}f_2(\mathbf{W}^T\mathbf{x})]|\mathbf{W}\}, \quad (7)$$

where $\mathbf{1} = [1, \ldots, 1]^T$ is an L-vector having ones as its elements and E is the expectation operator. The even, nonnegative, continuously differentiable cost function $f_1(t)$ has its only minimum at $t = 0$ and $f_1(t_1) \leq f_1(t_2)$ if $|t_1| < |t_2|$. If $f_1(t) = t^2$, $J(\mathbf{W})$ coincides with the usual mean-square error $E\{\|\mathbf{e}\|^2|\mathbf{W}\}$. In fact, one can always define a function $h_1(t) = \sqrt{f_1(t)}$ and express eqn (7) in the form $E\{\|h_1(\mathbf{e})\|^2|\mathbf{W}\}$. Denoting the jth components of the vectors $\mathbf{e}$, $\mathbf{x}$, and $\mathbf{w}(i)$ by ε_j, ξ_j, and $\omega_j(i)$, respectively, the error criterion (7) can be expressed as the sum of componentwise error criterions:

$$J(\mathbf{W}) = \sum_{j=1}^{L} E\{f_1(\varepsilon_j)|\mathbf{W}\}, \tag{8}$$

where

$$\varepsilon_j = \xi_j - \sum_{i=1}^{M} f_2[\mathbf{x}^T\mathbf{w}(i)]\omega_j(i). \tag{9}$$

In the following, we derive a stochastic gradient algorithm for minimizing the error criterion $J(\mathbf{W})$. In the derivation, the expectations may be omitted because they will be replaced by their instantaneous values in the final algorithm. Denoting the derivatives of $f_1(t)$ and $f_2(t)$ with respect to t by $g_1(t)$ and $g_2(t)$, respectively, we get from eqn (8)

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}(m)} = \sum_{j=1}^{L} g_1(\varepsilon_j)\frac{\partial \varepsilon_j}{\partial \mathbf{w}(m)}$$

and from eqn (9)

$$\frac{\partial \varepsilon_j}{\partial \mathbf{w}(m)} = -\mathbf{x}g_2[\mathbf{x}^T\mathbf{w}(m)]\omega_j(m) - f_2[\mathbf{x}^T\mathbf{w}(m)][0, \ldots, 0, 1, 0, \ldots, 0]^T,$$

where in the last vector the jth element is 1. Combining these expressions yields

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}(m)} = -\sum_{j=1}^{L} g_1(\varepsilon_j)\mathbf{x}g_2[\mathbf{x}^T\mathbf{w}(m)]\omega_j(m) - f_2[\mathbf{x}^T\mathbf{w}(m)]\sum_{j=1}^{L}[0, \ldots, 0, g_1(\varepsilon_j), 0, \ldots, 0]^T.$$

The last summation equals to $g_1(\mathbf{e})$ and $\sum_{j=1}^{L} g_1(\varepsilon_j)\omega_j(m) = g_1(\mathbf{e}^T)\mathbf{w}(m)$, so that

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}(m)} = -\mathbf{x}g_1(\mathbf{e}^T)\mathbf{w}(m)g_2[\mathbf{x}^T\mathbf{w}(m)] - f_2[\mathbf{x}^T\mathbf{w}(m)]g_1(\mathbf{e}).$$

Updating the weight vectors in the direction of negative gradient:

$$\mathbf{w}_{k+1}(m) = \mathbf{w}_k(m) - \mu_k\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}(m)}$$

yields the gradient descent algorithm

$$\mathbf{w}_{k+1}(m) = \mathbf{w}_k(m) + \mu_k\{g_1(\mathbf{e}_k^T)\mathbf{w}_k(m)g_2[\mathbf{x}_k^T\mathbf{w}_k(m)]\mathbf{x}_k + f_2[\mathbf{x}_k^T\mathbf{w}_k(m)]g_1(\mathbf{e}_k)\}, \tag{10}$$

$$\mathbf{e}_k = \mathbf{x}_k - \mathbf{W}_k f_2(\mathbf{W}_k^T\mathbf{x}_k) \tag{11}$$

for computing the weight vectors $\mathbf{w}(1), \ldots, \mathbf{w}(M)$ that minimize $J(\mathbf{W})$. It is seen that the new estimates are computed as weighted sums of the current input vector $\mathbf{x}_k$ and the nonlinear error vector $g_1(\mathbf{e}_k)$. Let $\mathbf{G}_2(\mathbf{x}_k^T\mathbf{W}_k)$ be the diagonal matrix whose ith element is $g_2(\mathbf{x}_k^T\mathbf{w}_k(i))$. Then eqn (10) can be written in matrix form as

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{x}_k g_1(\mathbf{e}_k^T)\mathbf{W}_k\mathbf{G}_2(\mathbf{x}_k^T\mathbf{W}_k) + g_1(\mathbf{e}_k)f_2(\mathbf{x}_k^T\mathbf{W}_k)]. \tag{12}$$

2.2. Special Cases

The update formula derived above is somewhat complex because it depends generally on three nonlinear functions $f_2(t)$, $g_1(t)$, and $g_2(t)$. Therefore, we consider some interesting special cases or approximations of eqn (10) that lead to simpler gradient algorithms.

1) If $f_2(t) = t$, or the coefficients in eqn (5) are linear, we get

$$\mathbf{w}_{k+1}(m) = \mathbf{w}_k(m) + \mu_k[g_1(\mathbf{e}_k^T)\mathbf{w}_k(m)\mathbf{x}_k + \mathbf{x}_k^T\mathbf{w}_k(m)g_1(\mathbf{e}_k)]. \tag{13}$$

In this case, the outputs $\mathbf{x}_k^T\mathbf{w}_k(m)$ of the neurons are linear, but the error criterion is generally defined by some nonquadratic function $f_1(t)$.

2) If the error in the previous case is quadratic, that is, $f_1(t) = t^2$, we can write the resulting algorithm compactly in matrix form as

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{x}_k\mathbf{e}_k^T\mathbf{W}_k + \mathbf{e}_k\mathbf{x}_k^T\mathbf{W}_k]. \tag{14}$$

The coefficient 2 from $g_1(t) = 2t$ has been absorbed to μ_k for convenience. This has earlier been proposed independently by Xu (1991) and Russo (1991), and is derived in Russo (1991) by minimizing the residual $J(\mathbf{W}) = \|(\mathbf{I} - \mathbf{W}\mathbf{W}^T)\mathbf{x}\|^2$. Inserting $\mathbf{e}_k = (\mathbf{I} - \mathbf{W}_k\mathbf{W}_k^T)\mathbf{x}_k$ into eqn (14) yields

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{x}_k\mathbf{x}_k^T(\mathbf{I} - \mathbf{W}_k\mathbf{W}_k^T)\mathbf{W}_k + (\mathbf{I} - \mathbf{W}_k\mathbf{W}_k^T)\mathbf{x}_k\mathbf{x}_k^T\mathbf{W}_k]. \tag{15}$$

From this one can see that the latter part of the update in eqn (14) is in fact the same as in eqn (1). In preliminary experiments (Xu, 1991), eqn (14) was found to be slightly more accurate than eqn (1), but otherwise these algorithms behaved very similarly. This is because the first update term $\mathbf{x}_k\mathbf{e}_k^T\mathbf{W}_k$ in eqn (14) is much less important than the latter one $\mathbf{e}_k\mathbf{x}_k^T\mathbf{W}_k$. The reason is that for each weight vector $\mathbf{w}(m)$ the update in the square brackets in eqn (14) becomes

$$\mathbf{e}_k^T\mathbf{w}_k(m)\mathbf{x}_k + \mathbf{x}_k^T\mathbf{w}_k(m)\mathbf{e}_k = (\mathbf{e}_k + \mathbf{x}_k)^T\mathbf{w}_k(m)\mathbf{x}_k - \mathbf{x}_k^T\mathbf{w}_k(m)\sum_{i=1}^{M}\mathbf{x}_k^T\mathbf{w}_k(i)\mathbf{w}_k(i).$$

Thus, the first term $\mathbf{e}_k^T\mathbf{w}_k(m)\mathbf{x}_k$ affects the coefficient of $\mathbf{x}_k$ only in the update. The coefficients of the weight

vectors $\mathbf{w}_k(i)$ depend solely on the latter term $\mathbf{x}_k^T\mathbf{w}_k(m)\mathbf{e}_k$. The first term can be omitted if the error $\mathbf{e}_k$ is small compared to $\mathbf{x}_k$, as it should be after initial learning phase. These considerations show that Oja's symmetric linear algorithm (1) is obtained from the general criterion (7) as a relevant approximation of the quadratic-linear special case.

3) Consider now the more general case in which $f_2(t)$ is nonlinear and the error criterion is quadratic, or $f_1(t) = t^2$. The general algorithm (12) simplifies then to the form

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{x}_k\mathbf{e}_k^T\mathbf{W}_k\mathbf{G}_2(\mathbf{x}_k^T\mathbf{W}_k) + \mathbf{e}_k f_2(\mathbf{x}_k^T\mathbf{W}_k)]. \quad (16)$$

For each weight vector $\mathbf{w}(m)$, the first term in the square brackets is again proportional to $\mathbf{x}_k$ only. Omitting it on the same grounds as before and inserting $\mathbf{e}_k$ from eqn (11), we get

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{x}_k - \mathbf{W}_k f_2(\mathbf{W}_k^T\mathbf{x}_k)]f_2(\mathbf{x}_k^T\mathbf{W}_k).$$

But this is exactly the algorithm (4) with the difference in notation that in (4) $f_2(t)$ is denoted by $g(t)$. This result helps greatly in understanding what the algorithm (4) is actually doing: it is an approximative stochastic gradient algorithm for minimizing the mean-square representation error $J(\mathbf{W}) = E\{\|\mathbf{x} - \mathbf{W}f_2(\mathbf{W}^T\mathbf{x})\|^2\}$. We have derived eqn (16) in an alternative way by expanding this error and computing its gradient directly, but it turns out easier to start from eqns (8) and (9). Because of the matrix $\mathbf{G}_2(\mathbf{x}_k^T\mathbf{W}_k)$, eqn (4) is not always as good an approximation to eqn (16) as eqn (1) is to eqn (14). It still holds that the term $\mathbf{e}_k f_2(\mathbf{x}_k^T\mathbf{W}_k)$ is much more important in learning than the omitted term $\mathbf{x}_k\mathbf{e}_k^T\mathbf{W}_k\mathbf{G}_2(\mathbf{x}_k^T\mathbf{W}_k)$. In the experiments, eqns (4) and (16) gave, in most cases, almost similar results, but sometimes eqn (16) performed better.

There are other ways of simplifying the general algorithm (12), for example, one can choose $g_1(t) = f_2(t)$ or omit the first term in the square brackets in eqns (12) or (13) as well. We do not discuss these possibilities more closely here.

2.3. Feature Extraction Using Nonlinear Criterion

Consider now another statistical optimization criterion defined by

$$J(\mathbf{W}) = \sum_{i=1}^{M} E\{f_1[\mathbf{x}^T\mathbf{w}(i)]|\mathbf{w}(i)\} + \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{M}\lambda_{ij}[\mathbf{w}(i)^T\mathbf{w}(j) - a_{ij}]. \quad (17)$$

The task is to maximize the first sum of conditional expectations that depend nonlinearly on the outputs $\mathbf{x}^T\mathbf{w}(i)$ of the neurons. Assuming that the number of neurons M is smaller than the dimension L of the vectors $\mathbf{x}$ and $\mathbf{w}(i)$, this can be regarded as a feature extraction problem where the function $f_1(t)$ measuring the performance of the network is generally nonquadratic. The function $f_1(t)$ satisfies similar conditions as earlier. The double summation imposes via the Lagrange multipliers $\lambda_{ij} = \lambda_{ji}$ constraints $\mathbf{w}(i)^T\mathbf{w}(j) = a_{ij}$ that are necessary for keeping the optimal solution bounded. The criterion (17) can be written more compactly

$$J(\mathbf{W}) = \mathbf{1}^T E[f_1(\mathbf{W}^T\mathbf{x})|\mathbf{W}] + \frac{1}{2}\mathrm{tr}[\mathbf{\Lambda}(\mathbf{W}^T\mathbf{W} - \mathbf{A})], \quad (18)$$

where tr denotes the trace of a matrix. The elements of the matrix $\mathbf{\Lambda}$ are λ_{ij} and the symmetric, nonsingular $M \times M$ matrix $\mathbf{A}$ has a_{ij} as its elements.

Because $f_1(t)$ is assumed to be continuously differentiable, the operations of partial differentiation and integration can be interchanged. The gradient of eqn (17) with respect to the weight vector $\mathbf{w}(m)$ is then

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{w}(m)} = E\{\mathbf{x}g_1[\mathbf{x}^T\mathbf{w}(m)]|\mathbf{w}(m)\} + \sum_{i=1}^{M}\lambda_{im}\mathbf{w}(i).$$

At the optimum, the gradients must vanish for $m = 1, \ldots, M$. The resulting equations can be written in matrix form as follows:

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = E[\mathbf{x}g_1(\mathbf{x}^T\mathbf{W})|\mathbf{W}] + \mathbf{W}\mathbf{\Lambda} = \mathbf{0}. \quad (19)$$

Differentiation of $J(\mathbf{W})$ with respect to the Lagrange multipliers yields the constraint equations

$$\mathbf{W}^T\mathbf{W} = \mathbf{A}. \quad (20)$$

Equations (19) and (20) are the necessary conditions for the optimum. Multiplying eqn (19) by $\mathbf{W}^T$ from left and taking into account eqn (20) yields

$$\mathbf{\Lambda} = -\mathbf{A}^{-1}\mathbf{W}^T E[\mathbf{x}g_1(\mathbf{x}^T\mathbf{W})|\mathbf{W}]. \quad (21)$$

Using this optimum value of $\mathbf{\Lambda}$ in the left hand side of eqn (19) yields

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = [\mathbf{I} - \mathbf{W}\mathbf{A}^{-1}\mathbf{W}^T]E[\mathbf{x}g_1(\mathbf{x}^T\mathbf{W})|\mathbf{W}]. \quad (22)$$

This can be used for searching the maximum of eqn (18) in the gradient algorithm

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k\frac{\partial J}{\partial \mathbf{W}}(\mathbf{W}_k). \quad (23)$$

Replacing the expectation by its instantaneous estimate $\mathbf{x}_k g_1(\mathbf{x}_k^T\mathbf{W}_k)$ leads to the gradient ascent algorithm

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu_k[\mathbf{I} - \mathbf{W}_k\mathbf{A}^{-1}\mathbf{W}_k^T]\mathbf{x}_k g_1(\mathbf{x}_k^T\mathbf{W}_k). \quad (24)$$

The constraint eqn (20) describes a general case in which the weight vectors $\mathbf{w}(m)$ of the neurons are linearly independent, but need not be mutually orthogonal and normalized to unity. By imposing a strict orthonormality condition $\mathbf{A} = \mathbf{I}$ we get rid of the inverse matrix $\mathbf{A}^{-1}$, and eqn (24) simplifies to eqn (3) with $g_1(t) = g(t)$. Another special case that yields a relatively

simple algorithm is that $\mathbf{\Lambda}$ is diagonal, which means that the weight vectors must be mutually orthogonal but generally not normalized to unity.

We have thus shown that eqn (3) is a stochastic gradient algorithm for finding the maximum of the objective function $\mathbf{1}^T E[f_1(\mathbf{W}^T\mathbf{x})|\mathbf{W}] = E\{\|h_1(\mathbf{W}^T\mathbf{x})\|^2|\mathbf{W}\}$ under the orthonormality constraints $\mathbf{W}^T\mathbf{W} = \mathbf{I}$. The algorithm is somewhat approximative in the sense that the expression of $\mathbf{\Lambda}$ is determined from the optimal solution and used then everywhere.

The same derivation yields eqn (1), too. In this case $f_1(t) = 0.5t^2$, and the criterion function to be maximized under the constraint $\mathbf{W}^T\mathbf{W} = \mathbf{I}$ takes the form $J(\mathbf{W}) = \mathrm{tr}(\mathbf{W}^T\mathbf{R}_{xx}\mathbf{W})$, where $\mathbf{R}_{xx} = E(\mathbf{x}\mathbf{x}^T)$ is the correlation matrix of the input vectors. Clearly, the optimal solution of this maximization problem is such that the columns of $\mathbf{W}$ must be some orthonormal basis vectors spanning the M-dimensional PCA subspace of the input vectors. The optimal matrix $\mathbf{W}$ is not unique, but the PCA subspace spanned by its columns is. Stated in another way, the projection matrix $\mathbf{W}\mathbf{W}^T$ onto the PCA subspace is unique.

Several authors (Baldi & Hornik, 1991; Hertz et al., 1991; Hrycej, 1992; Karhunen, 1982; Oja, 1989; Oja, 1992; Williams, 1985; Xu, 1991) have justified both theoretically and experimentally that the symmetric algorithm (1) indeed converges to the M-dimensional PCA subspace of the input vectors. However, the only quite heuristic derivation available for eqn (1) until recently (Oja, 1992; Xu, 1991) has been to replace the weight vector $\mathbf{w}_k$ in Oja's single neuron learning rule (Oja, 1982) by the weight matrix $\mathbf{W}_k$. We have now ended up to eqn (1) in two ways: as an approximation of the gradient algorithm minimizing $J(\mathbf{W}) = E\{\|(\mathbf{I} - \mathbf{W}\mathbf{W}^T)\mathbf{x}\|^2|\mathbf{W}\}$, and from eqn (17) by choosing $f_1(t) = 0.5t^2$. The equivalence of these criteria can be established by noting that

$$E\{\|(\mathbf{I} - \mathbf{W}\mathbf{W}^T)\mathbf{x}\|^2|\mathbf{W}\} = E\{\|\mathbf{e}\|^2|\mathbf{W}\} = E\{\|\mathbf{x} - \hat{\mathbf{x}}\|^2|\mathbf{W}\}$$

where $\hat{\mathbf{x}} = \mathbf{W}\mathbf{W}^T\mathbf{x}$ is the estimate of $\mathbf{x}$. From the orthogonality principle of estimation theory it follows that the optimal estimate $\hat{\mathbf{x}}$ must satisfy $E\{\mathbf{e}^T\hat{\mathbf{x}}|\mathbf{W}\} = 0$, which yields $E\{\mathbf{x}^T\hat{\mathbf{x}}|\mathbf{W}\} = E\{\hat{\mathbf{x}}^T\hat{\mathbf{x}}|\mathbf{W}\}$ or $\mathbf{W}^T\mathbf{W} = \mathbf{I}$. But under this condition

$$J(\mathbf{W}) = E\{\|(\mathbf{I} - \mathbf{W}\mathbf{W}^T)\mathbf{x}\|^2|\mathbf{W}\} = E\{\|\mathbf{x}\|^2 - \|\mathbf{W}^T\mathbf{x}\|^2|\mathbf{W}\},$$

which is minimized when $E\{\|\mathbf{W}^T\mathbf{x}\|^2|\mathbf{W}\} = \mathrm{tr}(\mathbf{W}^T\mathbf{R}_{xx}\mathbf{W})$ is maximized. This shows the equivalence of the two criteria in the special case of linear learning. Observe also that if the optimality condition $\mathbf{W}^T\mathbf{W} = \mathbf{I}$ is assumed to be valid, the first term in the square brackets in eqn (15) vanishes, and eqn (15) becomes the same as eqn (1).

In the comparable case of nonlinear learning $f_1(t) = t^2, f_2(t) \neq t$ in eqn (7), and $g_1(t)$ in eqn (18) equals to $f_2(t)$ in eqn (7). Now the two criteria are no longer equivalent. They yield somewhat different algorithms, eqn (4) as an approximation of eqn (16), and eqn (3) as a special case $\mathbf{\Lambda} = \mathbf{I}$ of eqn (24).

3. PROPERTIES OF THE ALGORITHMS

3.1. General Remarks

The general algorithm (10)–(11) can be realized using the network shown in Figure 1. Note that the update formula (10) depends on the weight vectors of the other neurons through the error vector (11) only. This vector is the same for all the neurons and can be computed first. The update (10) itself is local, depending on the error vector $\mathbf{e}_k$, input vector $\mathbf{x}_k$, and the weight vector $\mathbf{w}_k(m)$ of the neuron to be updated. The special cases considered in Subsection 2.2. allow usually a simpler realization. The network is completely symmetric, which has certain advantages and drawbacks as discussed in the Introduction.

One can easily construct the averaged differential equation corresponding to eqns (10)–(11). Denoting in the continuous case the weight vector of mth neuron at time t by $\mathbf{w}_m(t)$, one gets by forming $[\mathbf{w}_{k+1}(m) - \mathbf{w}_k(m)]/\mu_k$ and letting $\mu_k \to 0$

$$\frac{d\mathbf{w}_m(t)}{dt} = -E\{\mathbf{x}g_1(\mathbf{e}^T)\mathbf{w}_m g_2(\mathbf{x}^T\mathbf{w}_m)|\mathbf{W}\} - E\{g_1(\mathbf{e})f_2(\mathbf{x}^T\mathbf{w}_m)|\mathbf{W}\}, \quad (25)$$

where $\mathbf{e}$ and $\mathbf{x}$ are also functions of time. On certain conditions (Oja, 1983), the stochastic algorithm (10) converges to one of the asymptotically stable roots of eqn (25). These roots are difficult to solve even in the single neuron case, except for the linear algorithm (15). For determining the roots, one should generally know the distribution of $\mathbf{x}$ and resort then to numerical techniques (Oja et al., 1991).

An advantage of the optimization approach is that the criterion functions (7) and (18) yield useful information on the properties of the derived algorithms and help to explain the effect of various choices. First, we see that due to the complete symmetry the optimal weight matrix $\mathbf{W}^*$ is not unique. This is because its column vectors that are the weight vectors $\mathbf{w}^*(m)$ of the neurons can always be interchanged without changing the value of the criterion function $J(\mathbf{W}^*)$. Thus, initial values and other random factors determine the order of columns in $\mathbf{W}^*$. In the linear algorithms (1) and (15) there is even more freedom because the columns of $\mathbf{W}_k$ may converge to any orthonormal basis of the M-dimensional PCA subspace. In practice, the algorithm (1) usually converges to such weight vectors $\mathbf{w}^*(m)$ in the PCA subspace that produce roughly equal output variances (Hrycej, 1992). If it is important that

the neuron weight vectors have some order, there exist several possibilities for achieving this in the nonlinear case. One can, for example, iterate first with one neuron only and add a second neuron after some convergence has already taken place, and so on. It is also possible to use different learning rates μ_k for different neurons, or impose additional constraints that make the solution unique.

The criterion (18) and the algorithm (24), typically with $\mathbf{A} = \mathbf{I}$, are mainly useful in feature extraction. If $f_1(t) \neq t^2$, the extracted features $g_1[\mathbf{x}^T\mathbf{w}(m)]$ maximize other than the usual quadratic information measure. In eqn (7), $f_1(t)$ and $f_2(t)$ define, respectively, a possibly nonquadratic error measure and nonlinear coefficients in the approximation of x. This criterion and the resulting gradient algorithms can be used for achieving two related but somewhat different goals: 1) separation of the input signal or data x into its strongest subsignals or components $f_2[\mathbf{x}^T\mathbf{w}(i)]\mathbf{w}(i)$, or 2) extraction of the basis signals $\mathbf{w}(i)$ whose linear combinations the input vectors x primarily are.

3.2. Choice of the Nonlinearity

We consider next some possible alternatives for the functions $f_1(t)$ and $f_2(t)$ and their relative merits. Figure 2 shows some typical choices of the function $f_1(t)$. The dashed curve is the usual quadratic criterion $f_1(t) = t^2/2$, the dash–dot line is the linear criterion $f_1(t) = |t|$, and the solid line is the criterion function $f_1(t) = \text{lncosh}(t) = \ln 0.5(e^t + e^{-t})$. The corresponding derivatives $g_1(t) = t$, $g_1(t) = \text{sgn}(t)$, and $g_1(t) = \tanh(t)$, where $\text{sgn}(t)$ denotes the sign of t, have been depicted in Figure 3. The derivative $g_1(t)$ is a similar monotonic odd function as $f_2(t)$; Figure 3 represents typical choices of $f_2(t)$, too.

One could also consider a criterion function $f_1(t)$ that grows faster than quadratically [e.g., $f_1(t) = t^4/4$]. In practice, such a criterion is usually not good because

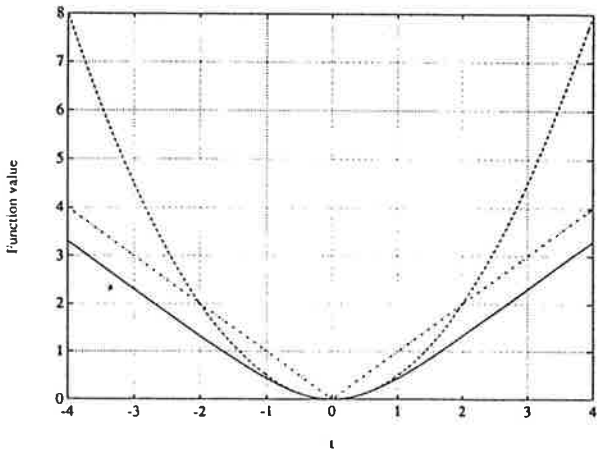

FIGURE 2. Some typical choices of the function $f_1(t)$. Dashed curve: $f_1(t) = t^2/2$; dash–dot line: $f_1(t) = |t|$; solid curve: $f_1(t) = \text{lncosh}(t)$.

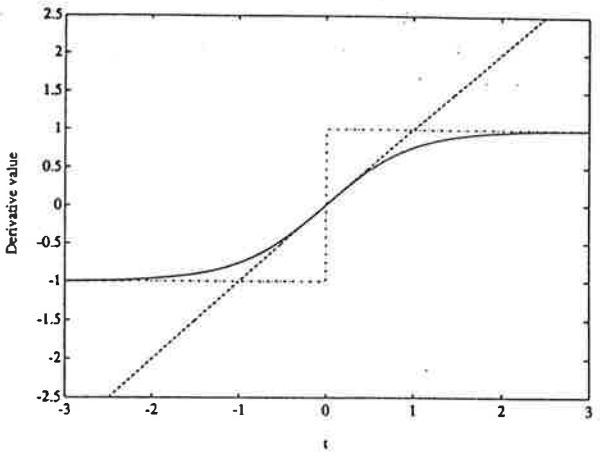

FIGURE 3. Derivatives $g_1(t) = t$ (dashed line), $g_1(t) = \text{sgn}(t)$ (dot–dash line), and $g_1(t) = \tanh(t)$ (solid curve) of the functions $f_1(t)$ depicted in Figure 2.

it tends to give too much weight to large values of t and leads more easily to stability problems in gradient algorithms. It is often preferable to choose $f_1(t)$ so that it grows less than quadratically with t. It is well known that optimization criteria based on absolute error are more robust against outliers and impulsive noise than those minimizing some quadratic error. In spite of this, standard approaches are based on quadratic criteria because they are mathematically easier to handle, leading typically to linear algorithms. The criterion function $f_1(t) = |t|$ is tempting because of its simplicity. However, its derivative $\text{sgn}(t)$ is not continuous and has only two possible values, $+1$ and -1. Therefore, $f_2(t) = \text{sgn}(t)$ leads to a very crude representation (5) of x and is not good in this respect.

From Figure 2 one can see that the criterion function $f_1(t) = \text{lncosh}(t)$ behaves at small values of $|t|$ roughly quadratically, and for larger values of $|t|$ $\text{lncosh}(t) \approx |t| - \ln 2$. Thus, $f_1(t) = \text{lncosh}(t)$ is satisfactory in many respects. The only problem is that its derivative $\tanh(t)$ is not always suitable to be used as the function $f_2(t)$ because its values are restricted to the interval $[-1, 1]$. This difficulty can be alleviated to some extent at least by scaling the input vectors x suitably. According to our experience, often the best choice of $f_2(t)$ is a function that grows less than linearly but does not saturate. Examples of such functions are $f_2(t) = \text{sgn}(t)\sqrt{|t|}$ and $f_2(t) = \text{sgn}(t)\ln(1 + \alpha|t|)$, where α is a scaling constant. Some other functions used especially in robust statistics have been discussed in Cichocki and Unbehauen (1993).

3.3. Choice of the Gain Parameter

In practice, it is important to choose the gain parameter μ_k so that the algorithm converges well and is stable (i.e., the weight vectors remain bounded). Typically, too large values of μ_k give rise to instability, whereas

too a conservative choice of μ_k leads to possibly intolerably slow though smooth convergence. We have recently derived for the basic symmetric algorithm (1) the following stability result, which extends the analysis outlined by Karhunen & Joutsensalo (1991b).

THEOREM 1. *The algorithm* (1) *is stable if the gain parameter* μ_k *satisfies at every iteration the condition*

$$0 \le \mu_k \le 2\|\mathbf{x}_k\|^{-2}, \tag{26}$$

and the initial weight matrix $\mathbf{W}_1$ *is chosen so that the largest eigenvalue of* $\mathbf{W}_1\mathbf{W}_1^T$ *is at most* 2. *If the largest eigenvalue of the matrix* $\mathbf{W}_1\mathbf{W}_1^T$ *or more generally* $\mathbf{W}_k\mathbf{W}_k^T$ *is* $\lambda_1 > 2$, μ_k *must satisfy the condition*

$$0 \le \mu_k \le \frac{2}{(\lambda_1 - 1)\|\mathbf{x}_k\|^2}. \tag{27}$$

The proof is somewhat lengthy and not directly related to the nonlinear algorithms. Therefore, it will be presented elsewhere together with other stability considerations. However, we have made some simple experiments with eqn (1) that are in excellent agreement with the condition (26).

Theorem 1 can be generalized for the nonlinear algorithm (3) as follows.

COROLLARY 1. *The algorithm* (3) *is stable if the conditions of Theorem* 1 *hold and* $|g(t)| \le |t|$ *for all* t, *that is, the odd function* $g(t)$ *grows at most linearly.*

This result can be established rather easily by noting that for each individual weight vector the update formula (3) differs from (1) only in that the nonlinear function $g(t)$ is applied to the inner product $\mathbf{x}_k^T\mathbf{w}_k(m)$.

If the function $g(t)$ is chosen so that it grows less than linearly [e.g., $g(t) = \tanh(t)$ or $g(t) = \mathrm{sgn}(t)\ln(1 + |t|)$], the stability properties of eqn (3) are in practice better than those of eqn (1). The algorithm can be made stable also for functions $g(t)$ growing faster than linearly [e.g., $g(t) = t^3$] if μ_k is kept sufficiently small. However, it is difficult to determine a proper value for μ_k in this case, and stability problems are more likely to occur. Much the same remarks seem to hold for the other nonlinear algorithms in practice.

In algorithms like eqn (1), the fastest initial convergence rate is usually achieved when μ_k is roughly in the middle of its stability region. However, the variance of the weight vector estimates is relatively large in this case. Therefore, μ_k should be made gradually smaller after initial convergence. In practical realization of the algorithms, it is often desirable to avoid division with $\|\mathbf{x}_k\|^2$. Then one can use a constant gain parameter μ that is, for example, 0.1–0.2 of the average upper bound in eqn (26) and diminish the value of μ_k later if necessary. These recommendations apply largely to the nonlinear algorithms discussed in this paper, too. Various strategies of choosing the gain parameter in stochastic gradient descent algorithms have been discussed in Darken, Chang, and Moody (1992).

3.4. Relationship to Higher-Order Statistics

The main justification of using nonlinear functions $f(t)$ and $g(t)$ in PCA type networks is that they introduce higher-order statistics into the computations. Consider, for example, the algorithm (4), which is obtained as an approximation of eqn (16). For each neuron weight vector $\mathbf{w}(m)$ it can be written

$$\mathbf{w}_{k+1}(m) = \mathbf{w}_k(m) + \mu_k\mathbf{e}_k g[\mathbf{x}_k^T\mathbf{w}_k(m)], \tag{28}$$

where $\mathbf{e}_k$ is defined in eqn (11) and $g(t) = f_2(t)$. Thus, in the update term the error vector $\mathbf{e}_k$ is correlated with the nonlinear output $g[\mathbf{x}_k^T\mathbf{w}_k(m)]$ of the neuron. Assuming that $g(t)$ can be expanded in Taylor series, one can see that higher than merely second-order moments (correlations) of the components of $\mathbf{x}_k$ enter into the computations. For example, $g(t) = \tanh(t)$ has the Taylor series expansion $g(t) = t - t^3/3 + 2t^5/15 - \ldots$ for $|t| < \pi/2$.

Generally, nonlinearities and higher-order moments tend to increase the independence of the output signals in PCA-type networks. The standard PCA networks and linear Hebbian learning algorithms utilize only second-order statistics, producing uncorrelated output signals. Recently, Jutten and Herault (1991) together with Comon (1991) have introduced so-called INCA (independent component analysis) as an interesting counterpart to PCA [see also Cichocki & Unbehauen (1993)]. As its name implies, in INCA the goal is to decompose a signal into its statistically independent components (or as independent subsignals as possible). INCA is in many cases a much more meaningful decomposition than PCA because it aims at extracting the original signals from their mixture. In PCA, the uncorrelated output signals are almost always some linear combinations of the original source signals that form the mixture input signal. A drawback of INCA is that it is generally far more difficult to compute than PCA. It is not easy to verify independence exactly, because one should know or estimate the associated probability densities. Higher-order moments are required even in the case where the mixture is a linear combination of the original signals (Burel, 1992).

INCA is related to the so-called blind separation problem, which is becoming increasingly important in signal processing. In blind separation, the goal is to extract the original signals (sources) from their mixture when only little if any information on the sources themselves is available. In fact, some more or less neural approaches to blind separation have already been proposed (Burel, 1992; Jutten & Herault, 1991). However, these algorithms require generally inversion of a matrix at every iteration or are otherwise computationally complex.

The algorithms derived here are simpler in this sense. Our earlier experiments (Karhunen & Joutsensalo, 1992a,b) have shown that even eqn (4) can in some cases separate the source signals from their mixture. We do not claim that the algorithms derived in this paper perform INCA or solve the blind separation problem generally for linear mixtures, but they are clearly a step to this direction from PCA. One could argue that the ultimate goal should be to decompose an input signal into component signals in a meaningful way, not necessarily always INCA. The problem with this is that a signal can be decomposed into its subsignals in an infinite number of ways (Cichocki & Unbehauen, 1993), and some sensible assumptions or constraints, for example, independence, uncorrelatedness, and maximization of output variance are necessary. The approach presented in this paper starts from the optimization criterion (7) [or eqn (18)], where the form of the expansion (6) and the functions $f_1(t)$ and $f_2(t)$ impose the necessary constraints.

4. EXPERIMENTAL RESULTS

In this section, we study experimentally the performance of various algorithms in two cases. The first one is a simple but illustrative example adopted from Burel (1992). In the second case, the input signal is a noisy mixture of sinusoids. We consider both estimation of the frequencies of the sinusoids and their separation from the mixture.

4.1. Uniformly Distributed Independent Signals

In this case, the input vectors $\mathbf{x}_k$ are uniformly distributed inside the parallelogram shown in Figure 4. They can be generated using the formula

$$\mathbf{x}_k = \alpha_1 \mathbf{i}_1 + \alpha_2 \mathbf{i}_2, \tag{29}$$

where α_1 and α_2 are independent random numbers distributed uniformly over the interval $(-0.7, 0.7)$. The vectors $\mathbf{i}_1 = [-1/\sqrt{5}, 2/\sqrt{5}]^T$ and $\mathbf{i}_2 = [1, 0]^T$ are the basis vectors of INCA; they have been depicted in Figure 4 by solid lines. The corresponding PCA basis vectors $\mathbf{e}_1 = [3/\sqrt{13}, -2/\sqrt{13}]^T$ and $\mathbf{e}_2 = [2/\sqrt{13}, 3/\sqrt{13}]^T$ have been shown using dashed lines. The signs of the basis vectors $\mathbf{e}_i$ and $\mathbf{i}_i$ can be chosen arbitrarily.

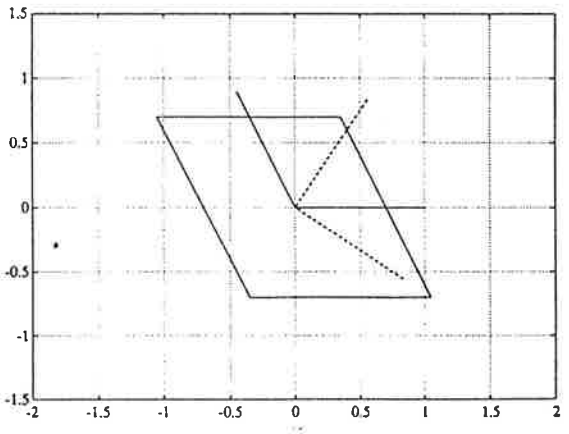

FIGURE 4. The parallelogram bounding the uniform distribution and the theoretical basis vectors of INCA (solid lines) and PCA (dashed lines) for this distribution.

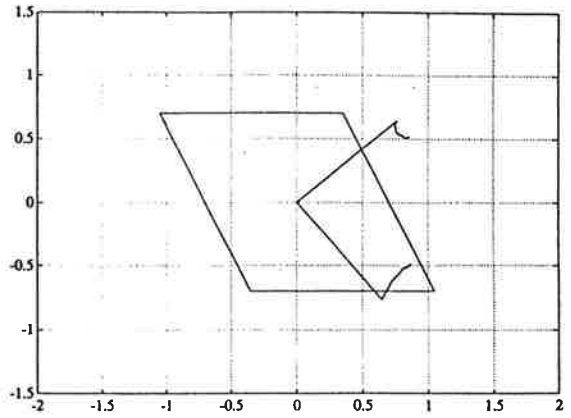

FIGURE 5. Learning curves and the final basis vectors (straight lines) for the algorithm (1).

The PCA basis vectors are always mutually orthonormal, and the coefficients $\mathbf{x}_k^T \mathbf{e}_i$ in the approximation

$$\hat{\mathbf{x}}_k = \sum_{i=1}^{M} (\mathbf{x}_k^T \mathbf{e}_i)\mathbf{e}_i \tag{30}$$

are uncorrelated. If $\mathbf{e}_1, \ldots, \mathbf{e}_M$ are the true principal eigenvectors of the data covariance matrix, the first coefficient $\mathbf{x}_k^T \mathbf{e}_1$ has the largest variance, the second coefficient second largest variance, etc. We note that this property does not hold for eqn (1), which estimates some orthonormal basis of the PCA subspace only. However, the PCA coefficients, or the coordinates of the data vectors $\mathbf{x}_k$ in the PCA coordinate system, are generally not independent. This is the case in Figure 4, too. Contrary to the PCA, the INCA basis vectors are usually not orthonormal, but the projections of the data vectors onto the INCA basis vectors are mutually independent. However, INCA is more difficult to use in approximating or representing the data vectors because nonorthogonal projections must be computed. The INCA formula corresponding to eqn (30) is

$$\hat{\mathbf{x}}_k = \mathbf{S}(\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T\mathbf{x}_k, \tag{31}$$

where $\mathbf{S} = [\mathbf{i}_1, \ldots, \mathbf{i}_M]$. Note that if the matrix $\mathbf{S}$ consists of PCA basis vectors, eqn (31) simplifies to eqn (30), because then $\mathbf{S}^T\mathbf{S} = \mathbf{I}$. From the definition of the expansion (5) it follows directly that

$$\hat{\mathbf{x}}_k = \sum_{i=1}^{M} f_2[\mathbf{x}_k^T \mathbf{w}(i)]\mathbf{w}(i), \tag{32}$$

if the algorithms (16) or (4) with $g(t) = f_2(t)$ are used. Thus, no matrix inversion is needed, and $\hat{\mathbf{x}}_k$ can be

computed relatively simply from the outputs $f_2[\mathbf{x}_k^T\mathbf{w}(i)]$ of the neurons and the respective weight vectors $\mathbf{w}(i)$.

We have tested the algorithms (1), (4), and (16) using the data described above. Figures 5, 6, and 7 show the results of a typical simulation. The number of data samples was 500. The gain parameter was held constant ($\mu = 0.05$) during the initial convergence, and then decreased slowly to zero. In eqns (4) and (16), the function $f_2(t)$ was $\mathrm{sgn}(t)\ln(1 + 20|t|)$. The figures show the initial values of the weight vectors, their trajectories during learning, and the final weight vectors marked by the straight lines starting from the origin. Both eqn (4) and eqn (16) yield a weight vector that is relatively close to the vector $\mathbf{i}_1$, eqn (16) being slightly better in this respect. On the other hand, the other weight vector in eqn (4) is slightly closer to the second INCA basis vector $\mathbf{i}_2$. The weight vectors given by eqn (1) are in this case something between INCA and PCA basis vectors.

Other values of α_i and $\mathbf{i}_i$ than those mentioned below eqn (29) have been tried, too. Sometimes a small third component was added to the data vectors for preventing the representation error going to zero, but this had very little effect on the results. General conclusions from these experiments are the following:

1. If the gain parameter μ_k is not too small and the function $f_2(t) = g(t)$ is chosen suitably, the nonlinear algorithms (16) and (4) converge usually roughly to the same weight vectors irrespective of the initial values and input data realization used. A different solution may occur especially if the initial values are poor and/or μ_k is too small. The algorithm (4) is somewhat more robust in this sense.
2. One of the weight vectors found by eqns (4) and (16) is usually close to the direction of the first INCA basis vector $\mathbf{i}_1$. The other weight vector lies in a roughly but not exactly orthogonal direction.
3. The nonlinearity $f_2(t) = \mathrm{sgn}(t)\ln(1 + \alpha|t|)$ usually gave the best results, provided that α is matched to fit the norms of the data vectors. The function $f_2(t) = \tanh(\alpha t)$ yielded more variable results. Fast growing functions such as $f_2(t) = t^3$ were more difficult to use because of stability problems, and the results varied even more.
4. It is difficult to assess the superiority of eqns (4) and (16). Either of these algorithms can yield a weight vector closest to the first INCA basis vector $\mathbf{i}_1$. The algorithm (4) is simpler and seems to be more robust.

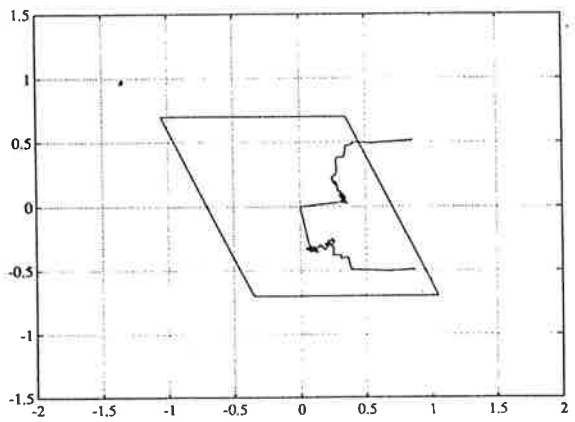

FIGURE 6. Learning curves and the final basis vectors (straight lines) for the algorithm (4).

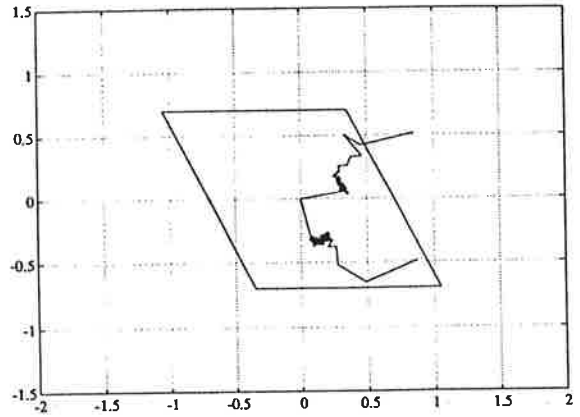

FIGURE 7. Learning curves and the final basis vectors (straight lines) for the algorithm (16).

4.2. Sinusoids in Noise

A more practical test example is estimation of the frequencies and/or separation of individual sinusoids from their noisy mixture. This problem arises naturally in several applications, and many different approaches of varying complexity and accuracy have been proposed (Kay, 1988; Therrien, 1992).

During the last years, some neural methods have been considered for sinusoidal frequency estimation or the related problem of estimating directions-of-arrival in array processing. Most of these try to optimize the maximum likelihood criterion function arising in this problem using, for example, backpropagation, simulated annealing, or Hopfield's net. This is often tedious or time consuming because the highly nonlinear criterion function has many local minima (Kay, 1988). A restriction of this approach is that noise should be white and Gaussian. The works of Kung, Diamantaras, and Taur (1991) and Russo (1991) are more closely related to ours because they use learning of sinusoidal frequencies as a test problem of proposed linear PCA subspace estimation algorithms.

An important class of methods for estimating sinusoidal frequencies is based on the notion of signal subspace (or its orthogonal complement) (Therrien, 1992). For noisy sinusoidal data, this is defined as the subspace spanned by pure sinusoidal vectors

$$\mathbf{e}_f = [1, e^{j2\pi f}, \ldots, e^{j2\pi f(L-1)}]^T \qquad (33)$$

at correct normalized frequencies $f \in [-0.5, 0.5]$; here $j = \sqrt{-1}$. For complex sinusoids, the correct dimensionality M of the signal subspace is naturally the same as the number J of the sinusoids in the data; for real-valued sinusoids $M = 2J$, because $\cos(2\pi f) = (e^{j2\pi f} + e^{-j2\pi f})/2$. Various signal subspace methods (Kay, 1988; Therrien, 1992) have become popular recently because they often provide good accuracy and are computationally clearly less demanding than the maximum likelihood approach. It can be shown theoretically that the signal subspace coincides with the M-dimensional PCA subspace of the input data. Here it is assumed that noise is white and that the PCA subspace is known exactly.

In practice, the signal subspace must be estimated somehow from the available data vectors. We use the proposed PCA type learning algorithms for this, and replace the unknown basis vectors of the signal subspace by the weight vectors $\mathbf{w}_k(i)$ of the neurons. For determining the frequencies, we use the well-known MUSIC method (Kay, 1988; Therrien, 1992), in which the sinusoidal frequencies are estimated by matching tentative vectors $\mathbf{e}_f$ to the signal subspace of correct dimensionality. In the general case of complex data, MUSIC frequency estimator then takes the form (Karhunen & Joutsensalo, 1991a, 1992b; Kay, 1988)

$$\hat{P}(f) = \frac{1}{L - \sum_{i=1}^{M} |\mathbf{e}_f^H \mathbf{w}(i)|^2}. \qquad (34)$$

Here H denotes the conjugate transpose. The estimated sinusoidal frequencies are obtained as peak locations of eqn (34). If the weight vectors are not roughly orthonormal, they must be orthonormalized explicitly prior to computing eqn (34) (Karhunen & Joutsensalo, 1991a).

One may question whether it is appropriate to use nonlinear PCA type learning algorithms in context with the MUSIC estimator, which relies on the theory of linear subspaces. Now observe that the representation (5) of the data vectors $\mathbf{x}$ is still linear with respect to the neuron weight vectors $\mathbf{w}(i)$. After convergence, these define a linear subspace that estimates the signal subspace of $\mathbf{x}$ because the nonlinear algorithms are derived by minimizing the statistical representation error (7). The nonlinear coefficients in eqn (5) are not needed in the MUSIC type estimator (34).

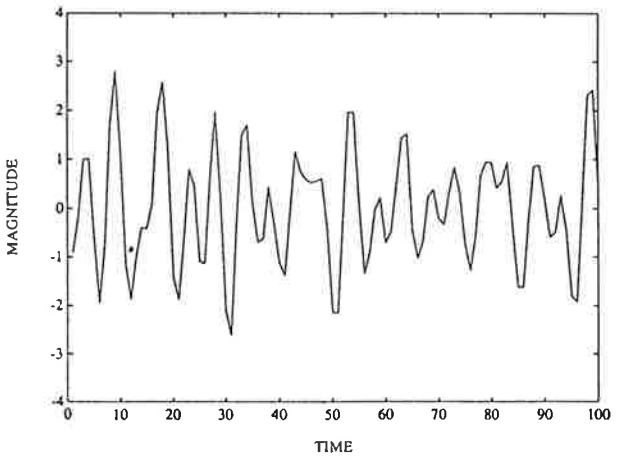

FIGURE 8. One hundred samples of the test data used in the example.

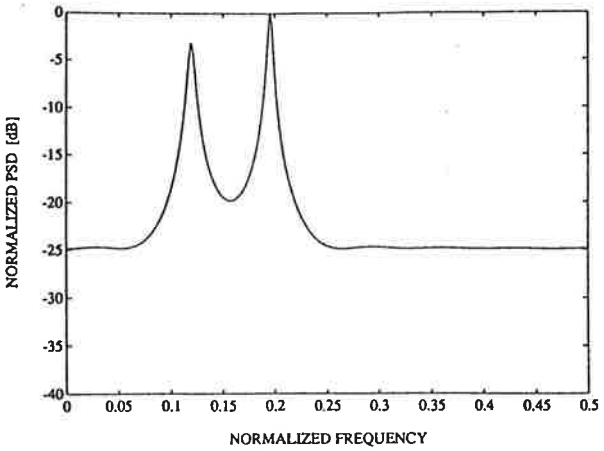

FIGURE 9. MUSIC estimator from eqn (1).

Our experimental data consists of N samples $x[1], \ldots, x[N]$ of J real sinusoids in additive noise:

$$x[k] = \sum_{m=1}^{J} A_m \cos(2\pi f_m k + \theta_m) + w[k]. \qquad (35)$$

The amplitudes A_m, frequencies f_m, and phases θ_m of the sinusoids are constants unknown to the learning algorithm. The data vectors

$$\mathbf{x}_k = (x[k], x[k+1], \ldots, x[k+L-1])^T \qquad (36)$$

are collected from L successive samples, and are used several times if necessary. The signal subspace dimension $M = 2J$ is usually the correct one.

We have tested different PCA-type algorithms with the above data using varying frequencies, types of noise, etc. First results on the linear algorithm (1) in white noise can be found in Karhunen and Joutsensalo (1991a,b). The nonlinear algorithms (2)–(4) have been studied using colored noise, too, in Karhunen and Joutsensalo (1992a,b). In the following, we present the most important conclusions from the experiments with these and the new nonlinear algorithms together with an example. The main attention is on eqns (4) and (16) because these algorithms have interesting signal separation properties.

1. In white Gaussian noise, all the algorithms yield reasonably good MUSIC estimators (34) after sufficient number of iterations (provided that the problem is not too difficult). If the signal-to-noise ratio is high, the basic algorithm (1) performs best, because PCA subspace is optimal in this situation.

2. In Karhunen and Joutsensalo (1991a,b), we used the data vectors $\mathbf{x}_k$ sequentially in eqn (1) and had difficulties in achieving high resolution. However, high resolution is possible, if $\mathbf{x}_k$'s are used in a random order. This diminishes the high correlation between subsequent data vectors, which causes convergence problems if the frequencies of the sinusoids are closely spaced. The algorithm (2) can also provide high resolution, whereas the more nonlinear algorithms, especially eqns (4) and (16), have difficulties with achieving it.
3. On the other hand, the most nonlinear algorithms (4) and (16) perform better than eqn (1) at low signal-to-noise ratios even in white Gaussian noise.
4. The nonlinear algorithms, in particular eqn (3), tolerate impulsive noise better than eqn (1) and are more stable provided that the nonlinearity grows less than linearly. This property follows from the definition of the criterion functions (17) and (7).
5. Especially eqns (4) and (16) have interesting filtering and separation properties. In most experiments with the functions $f_2(t) = g(t) = \text{sgn}(t)\ln(1 + \alpha|t|)$ or $\tanh(\alpha t)$, the outputs of individual neurons became almost pure sinusoids after learning, even though the input signal was a mixture of sinusoids in additive, possibly colored, noise. The algorithms (1), (2), or (15) with a linear Hebbian learning term $\mathbf{x}_k\mathbf{x}_k^T\mathbf{W}_k$ cannot usually separate the sinusoids in this way. An experimental comparison is presented below.
6. A comparison between eqns (4) and (16) reveals that eqn (16) has somewhat better separation properties, especially if the amplitudes of the sinusoids are different. In many cases, however, the two algorithms produce almost identical outputs of the neurons. On the other hand, for some reason the MUSIC frequency estimates given by eqn (4) are more accurate. Roughly speaking, this implies that eqn (4) has better noise filtering properties of these two algorithms.

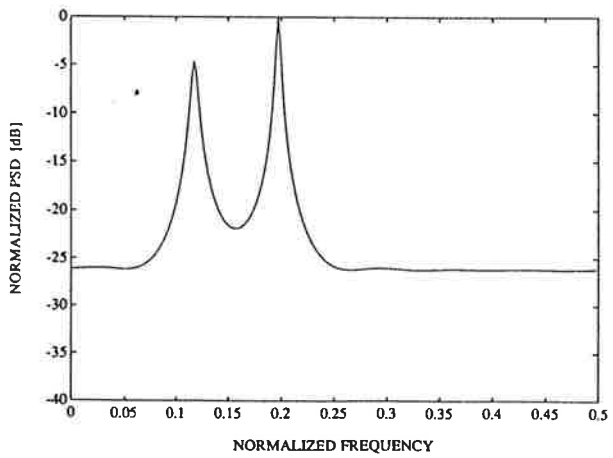

FIGURE 10. MUSIC estimator from eqn (4).

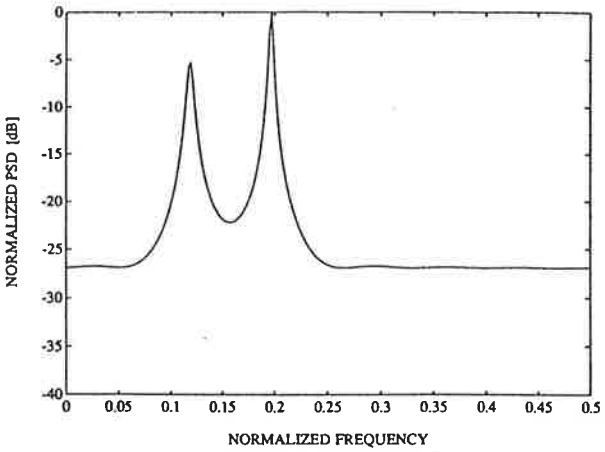

FIGURE 11. MUSIC estimator from eqn (16).

7. We have made some experiments with complex-valued data by replacing the transpose T in eqns (1)–(4) by the conjugate transpose H. After this minor change, these algorithms performed properly with sinusoidal data.
8. Tracking of slow changes in the frequencies is possible, but requires careful choice of the gain parameter μ_k. Some results with eqn (1) have been presented in Karhunen and Joutsensalo (1991a). The nonlinear algorithm (4) seems to possess somewhat better tracking properties than eqn (1).

EXAMPLE. The input data consisted of 100 samples of two sinusoids having normalized frequencies $f_1 = 0.11$ and $f_2 = 0.20$ and amplitudes $A_1 = 0.8$ and $A_2 = 1.2$, respectively. The data vectors had $L = 15$ components, and they were used 10 times for achieving convergence. In the algorithms (4) and (16), we used the nonlinearity $f_2(t) = g(t) = \text{sgn}(t)\ln(1 + 5|t|)$. The gain parameter was constant $\mu_k = 0.03$ during the first 300 iterations and then decreased slowly. Colored Gaussian noise $w(k)$ (SNR 5 dB) was generated from the autoregressive model $w(k) = 1.058w(k-1) - 0.81w(k-2) + v(k)$, where $v(k)$ is white Gaussian noise. The spectrum of this process has a disturbing peak frequency 0.15, because the poles of the filter corresponding to this AR model are $0.9\exp(\pm j2\pi 0.15)$. Figures 9, 10, and 11 show the MUSIC spectra (34) given by eqns (1), (4), and (16), respectively. The spectra are very similar, but the higher peak in Figures 10 and 11 is closer to the correct frequency 0.20.

After learning, test data (Figure 8) generated from another noise sequence realization using different phases of the sinusoids were inputted to the network. Figure 12A and B shows the outputs $\mathbf{x}_k^T\mathbf{w}_k(2)$ and $\mathbf{x}_k^T\mathbf{w}_k(4)$ of the second and fourth neuron of the linear network trained by eqn (1). The corresponding nonlinear outputs $f_2(t_2)$ and $f_2(t_4)$, $t_i = \mathbf{x}_k^T\mathbf{w}_k(i)$, of the networks trained using eqns (4) and (16) have been depicted in Figures 13 and 14, respectively. The signals

in Figures 13B and 14A have a clear fundamental frequency of about 0.20, and those in Figures 13A and 14B about 0.11, whereas the output signals of the linear network in Fig. 12A,B contain both frequencies in somewhat different proportions. The first and third neuron in each network produced qualitatively similar output signals as in Figures 12–14 with a phase difference. The curves appear somewhat rectangular because subsequent output samples have been connected by straight line segments.

Clearly, the nonlinear networks trained using eqns (4) and (16) have learned the sinusoids themselves and separated them into different output signals, whereas the linear network trained by eqn (1) cannot do this. The nonlinear networks have filtered some of the colored noise out, too. When the number of neurons was increased from four to six, each of the MUSIC estimators had three peaks roughly at frequencies 0.11, 0.15, and 0.20. In this case, two of the neurons trained by eqns (4) or (16) learned the sinusoidal frequency 0.11, two others the frequency 0.20, and the remaining

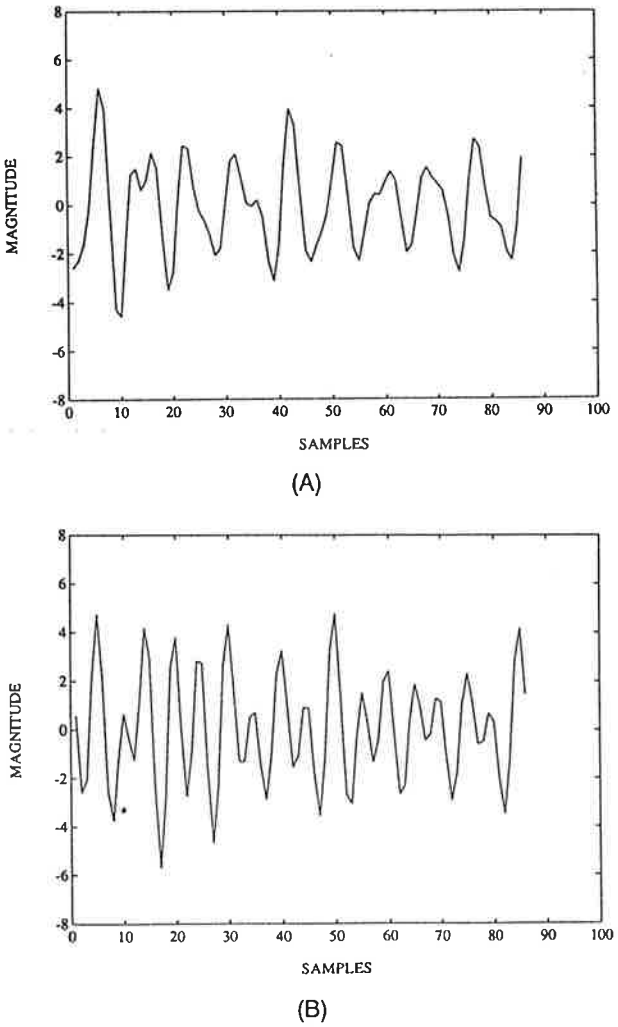

FIGURE 12. Outputs of the second (A) and fourth (B) neuron of the linear network trained using eqn (1) for the test data.

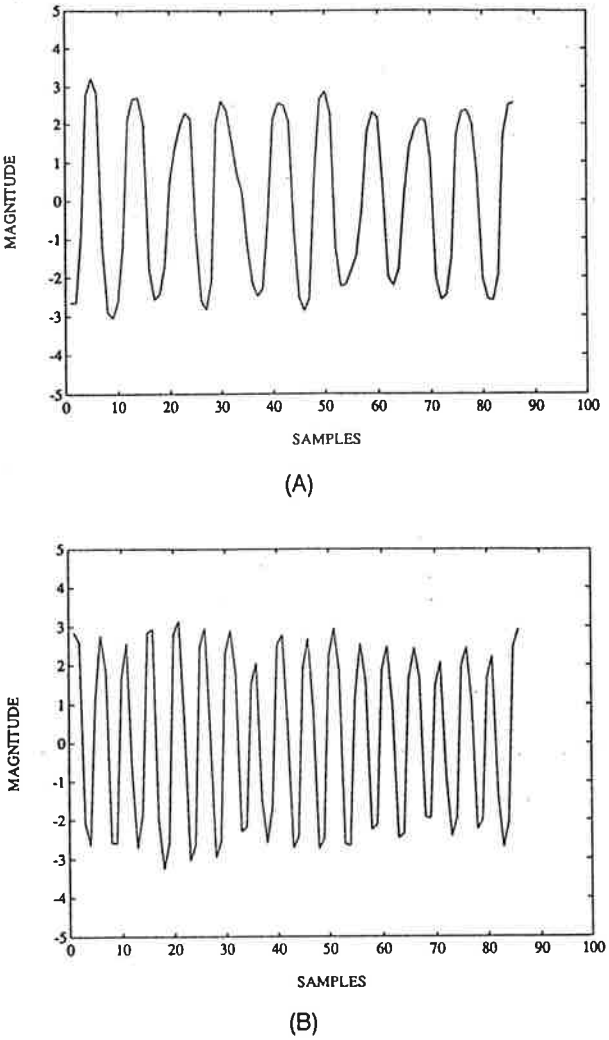

FIGURE 13. Outputs of the second (A) and fourth (B) neuron of the nonlinear network trained using eqn (4) for the test data.

two neurons produced an output corresponding roughly to the AR noise.

We also made a statistical test using 100 realizations of the above data. In each realization, the noise sequence and the phases of the sinusoids were chosen randomly. The frequencies of the sinusoids were estimated from eqn (34). The average absolute biases of the estimated frequencies f_1 and f_2 were, respectively: 0.0078 and 0.0029 for the linear algorithm (1); 0.0068 and 0.0020 for eqn (4); and 0.0085 and 0.0044 for eqn (16). If the function $f_2(t) = g(t) = \tanh(\alpha t)$ is used, the biases given by the nonlinear algorithms (4) and (16) are smaller, but the saturation effect typically clips off the highest values of the output signals. Some experiments with eqn (4) are presented in Karhunen and Joutsensalo (1992b).

5. CONCLUSIONS

In this paper, we have derived two classes of nonlinear PCA-type learning algorithms from the optimization

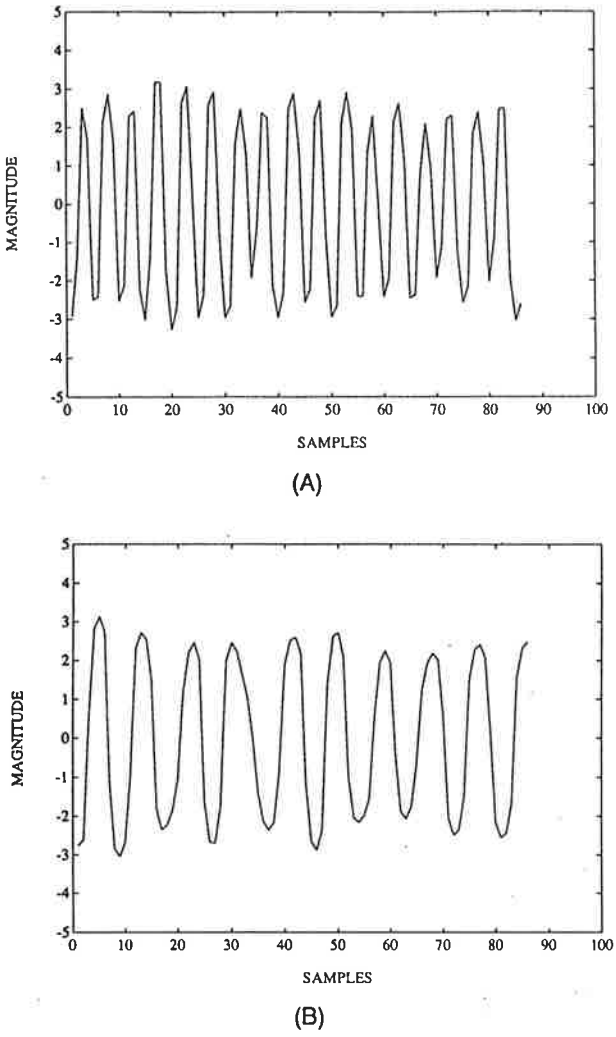

FIGURE 14. Outputs of the second (A) and fourth (B) neuron of the nonlinear network trained using eqn (16) for the test data.

criteria (7) and (17). Several of the existing linear or nonlinear PCA-type algorithms are obtained as special cases or approximations. The optimization criteria provide valuable information on the properties of the algorithms under various choices, and help to understand their mutual relationships. A suitably chosen nonlinearity makes the algorithm more robust against impulsive and colored noise. By taking into account higher-order statistics, some nonlinear algorithms are often able to separate component signals from their mixture. This is not possible with linear PCA subspace estimation algorithms. These properties are demonstrated experimentally using noisy sinusoidal data.

One could characterize the derived algorithms so that they produce results that lie somewhere between PCA and INCA. The standard PCA provides in the mean-square error sense optimal linear representation of the input signal and filters effectively white noise, but its separation capability is poor. The INCA algorithms are powerful in separating the component signals but they cannot filter noise and are not convenient for representing the data. The mildly nonlinear algorithms (3) and (13) are close to standard PCA subspace estimation algorithms, but are more robust against outliers and impulsive noise. The more nonlinear algorithms (4) and (16) are able to separate signals from their linear mixture provided that the component signals are not too much correlated. Furthermore, they can *simultaneously* filter noise and represent the original signal fairly well using a lower-dimensional subspace. Thus, they provide an interesting alternative to PCA and INCA.

REFERENCES

Baldi, P., & Hornik, K. (1991). Back-propagation and unsupervised learning in linear networks. In Y. Chauvin & D. Rumelhart (Eds.), *Back propagation: Theory, architecture and applications.* Hillsdale: Lawrence Erlbaum.

Burel, G. (1992). Blind separation of sources: A nonlinear neural algorithm. *Neural Networks,* **5,** 937–947.

Carlson, A. (1990). Anti-Hebbian learning in a non-linear neural network. *Biological Cybernetics,* **64,** 171–176.

Cichocki, A., & Unbehauen, R. (1993). *Neural networks for optimization and signal processing.* New York: J. Wiley.

Comon, P., Jutten, C., & Herault, J. (1991). Blind separation of sources, part II: Problems statement. *Signal Processing,* **24,** 11–20.

Darken, C., Chang, J., & Moody, J. (1992). Learning rate schedules for faster stochastic gradient search. In S. Y. Kung et al. (Eds.), *Neural networks for signal processing II* (pp. 3–12). New York: IEEE Press.

Földiak, P. (1990). Forming sparse representations by local anti-Hebbian learning. *Biological Cybernetics,* **64,** 165–170.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation.* Reading: Addison–Wesley.

Hrycej, T. (1992). Supporting supervised learning by self-organization. *Neurocomputing,* **4,** 17–30.

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

Karhunen, J. (1982). *On the recursive estimation of the eigenvectors of correlation type matrices.* Lic. tech. thesis (in Finnish), Helsinki University of Technology, Finland.

Karhunen, J. (1984). *Recursive estimation of eigenvectors of correlation type matrices for signal processing applications.* Dr. tech. dissertation, Helsinki University of Technology, Finland.

Karhunen, J., & Joutsensalo, J. (1991a). Tracking of sinusoidal frequencies by neural network learning algorithms. *Proceedings of the 1991 IEEE International Conference on Acoustics, Speech, and Signal Processing,* Toronto, Canada (pp. 3177–3180). New York: IEEE Press.

Karhunen, J., & Joutsensalo, J. (1991b). Frequency estimation by a Hebbian subspace learning algorithm. In T. Kohonen et al. (Eds.), *Artificial neural networks* (pp. 1637–1640). Amsterdam: North–Holland.

Karhunen, J., & Joutsensalo, J. (1992a). Nonlinear Hebbian algorithms for sinusoidal frequency estimation. In I. Aleksander and J. Taylor (Eds.), *Artificial neural networks,* 1–2 (pp. 1199–1102). Amsterdam: North–Holland.

Karhunen, J., & Joutsensalo, J. (1992b). Learning of sinusoidal frequencies by nonlinear constrained Hebbian algorithms. In S. Y. Kung et al. (Eds.), *Neural networks for signal processing II* (pp. 39–48). New York: IEEE Press.

Kay, S. M. (1988). *Modern spectral estimation: Theory and application.* Englewood Cliffs: Prentice–Hall.

Kotilainen, P., Saarinen, J., & Kaski, K. (1992). Hardware implementations of PCA neural networks. In I. Aleksander & J. Taylor (Eds.), *Artificial neural networks,* 2 (pp. 1427–1430). Amsterdam: North–Holland.

Kung, S. Y. (1993). *Digital neural networks.* Englewood Cliffs: Prentice–Hall.

Kung, S. Y., Diamantaras, K. I., & Taur, J. S. (1991). Neural networks for extracting pure/constrained/oriented principal components. In R. Vaccaro (Ed.), *SVD and signal processing, II* (pp. 57–81). New York: Elsevier.

Linsker, R. (1988). Self-organization in a perceptual network. *IEEE Computer,* **21,** 105–117.

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology,* **15,** 267–273.

Oja, E. (1983). *Subspace methods of pattern recognition.* Letchworth: Research Studies Press and J. Wiley.

Oja, E. (1989). Neural networks, principal components, and subspaces. *International Journal of Neural Systems,* **1,** 61–68.

Oja, E. (1992). Principal components, minor components, and linear neural networks. *Neural Networks,* **5,** 927–935.

Oja, E., Ogawa, H., & Wangviwattana, J. (1991). Learning in nonlinear constrained Hebbian networks. In T. Kohonen et al. (Eds.), *Artificial neural networks* (pp. 385–390). Amsterdam: North–Holland.

Oja, E., Ogawa, H., & Wangviwattana, J. (1992). Principal component analysis by homogeneous neural networks, part II: Analysis and extensions of the learning algorithms. *IEICE Transactions on Information and Systems* (Japan), E75-D, **3,** 376–382.

Russo, L. (1991). An outer product neural network for extracting principal components from a time series. In B. H. Juang et al. (Eds.), *Neural networks for signal processing* (pp. 161–170). New York: IEEE Press.

Sanger, T. D. (1991). Optimal hidden units for two-layer nonlinear feedforward neural networks. *International Journal of Pattern Recognition and Artificial Intelligence,* **5,** 545–561.

Shapiro, J. L., & Prügel–Bennett, A. (1992). Unsupervised Hebbian learning and the shape of the neuron activation function. In I. Aleksander & J. Taylor (Eds.), *Artificial neural networks,* 2 (pp. 179–182). Amsterdam: North–Holland.

Sirat, J. A. (1991). A fast neural algorithm for principal component analysis and singular value decomposition. *International Journal of Neural Systems,* **2,** 147–155.

Softky, W. R., & Kammen, D. M. (1991). Correlations in high dimensional or asymmetric data sets: Hebbian neuronal processing. *Neural Networks,* **4,** 337–347.

Taylor, J. G., & Coombes, S. (1993). Learning higher order correlations. *Neural Networks,* 6, 423–427.

Therrien, C. W. (1992). *Discrete random signals and statistical signal processing.* Englewood Cliffs: Prentice–Hall.

White, R. (1992). Competitive Hebbian learning: Algorithm and demonstrations. *Neural Networks,* **5,** 261–275.

Williams, R. (1985). Feature discovery through error-correcting learning. (Tech. Rep. 8501). San Diego: University of California, Institute of Cognitive Science.

Xu, L. (1991). Least MSE reconstruction for self-organization. *Proceedings of the International Joint Conference on Neural Networks,* Singapore, November 1991 (part I: pp. 2362–2367; part II: pp. 2368–2373).