Chapter 3
Neural Networks for Principal Component Analysis

3.1 Introduction

PCA is a statistical method, which is directly related to EVD and SVD. Neural networks-based PCA method estimates PC online from the input data sequences, which especially suits for high-dimensional data due to the avoidance of the computation of large covariance matrix, and for the tracking of nonstationary data, where the covariance matrix changes slowly over time. Neural networks and algorithms for PCA will be described in this chapter, and algorithms given in this chapter are typically unsupervised learning methods.

PCA has been widely used in engineering and scientific disciplines, such as pattern recognition, data compression and coding, image processing, high-resolution spectrum analysis, and adaptive beamforming. PCA is based on the spectral analysis of the second moment matrix that statistically characterizes a random vector. PCA is directly related to SVD, and the most common way to perform PCA is via the SVD of a data matrix. However, the capability of SVD is limited for very large data sets.

It is well known that preprocessing usually maps a high-dimensional space to a low-dimensional space with the least information loss, which is known as feature extraction. PCA is a well-known feature extraction method, and it allows the removal of the second-order correlation among given random processes. By calculating the eigenvectors of the covariance matrix of the input vector, PCA linearly transforms a high-dimensional input vector into a low-dimensional one whose components are uncorrelated.

X. Kong et al., *Principal Component Analysis Networks and Algorithms*,
DOI 10.1007/978-981-10-2915-8_3

PCA is often based on the optimization of some information criterion, such as the maximization of the variance of the projected data or the minimization of the reconstruction error. The aim of PCA is to extract m orthonormal directions $\overline{\overline{\boldsymbol{w}}}_i \in \Re^n, i = 1, 2, \ldots, m$, $m < n$, in the input space that account for as much of the data's variance as possible. Subsequently, an input vector $\boldsymbol{x} \in \Re^n$ may be transformed into a lower m-dimensional space without losing essential intrinsic information. The vector x can be represented by being projected onto the m-dimensional subspace spanned by w_i using the inner products $\boldsymbol{x}^{\mathrm{T}}\overline{\overline{\boldsymbol{w}}}_i$. This achieves dimensionality reduction.

PCA finds those unitary directions $\overline{\overline{\boldsymbol{w}}} \in \Re^n$, along which the projections of the input vectors, known as the principal components (PCs), $y = \boldsymbol{x}^{\mathrm{T}}\overline{\overline{\boldsymbol{w}}}$, have the largest variance $E_{\mathrm{PCA}}(\boldsymbol{w}) = E[y^2] = \overline{\overline{\boldsymbol{w}}}^{\mathrm{T}}\boldsymbol{C}\overline{\overline{\boldsymbol{w}}} = \overline{\overline{\boldsymbol{w}}}^{\mathrm{T}}\boldsymbol{C}\overline{\overline{\boldsymbol{w}}}/\|\boldsymbol{w}\|^2$, where $\overline{\overline{\boldsymbol{w}}} = \boldsymbol{w}/\|\boldsymbol{w}\|$. When $\boldsymbol{w} = \alpha\boldsymbol{c}_1$, $E_{\mathrm{PCA}}(\boldsymbol{w})$ take its maximum value, where α is a scalar. When $\alpha = 1, \boldsymbol{w}$ becomes a unit vector. By repeating maximization of $E_{\mathrm{PCA}}(\boldsymbol{w})$ but limiting w to be orthogonal to $\boldsymbol{c}_1$, the maximization of $E_{\mathrm{PCA}}(\boldsymbol{w})$ is equal to λ_2 at $\boldsymbol{w} = \alpha\boldsymbol{c}_2$. Following this deflation procedure, all the m principal directions $\overline{\overline{\boldsymbol{w}}}_i$ can be derived. The projections $y_i = \boldsymbol{x}^{\mathrm{T}}\overline{\overline{\boldsymbol{w}}}_i, i = 1, 2, \ldots, m$ are the PCs of $\boldsymbol{x}$. A linear least square (LS) estimate $\hat{\boldsymbol{x}}$ can be constructed for the original input $\boldsymbol{x}$ as $\hat{\boldsymbol{x}} = \sum_{i=1}^{m} a_i(t)\overline{\overline{\boldsymbol{w}}}_i$. As to other interpretations or analyses of PCA, see [1–4] for more details.

3.2 Review of Neural-Based PCA Algorithms

Neural networks on PCA pursue an effective "online" approach to update the eigen direction after each presentation of a data point, which are especially suitable for high-dimensional data and for the tracking of nonstationary data. In the last decades, many neural network-based PCA learning algorithms were proposed, among which, the Hebbian and Oja's learning rules are the bases. Overall, the existing neural network-based PCA algorithms can be grouped into the following classes: the Hebbian rule-based PCA algorithms, least mean squared error-based PCA algorithms, other optimization-based PCA algorithms, anti-Hebbian rule-based PCA algorithms, nonlinear PCA algorithms, constrained PCA algorithms, localized PCA algorithms, and other generalizations of the PCA. These algorithms will be analyzed and discussed in the above order.

3.3 Neural-Based PCA Algorithms Foundation

3.3.1 Hebbian Learning Rule

The classical Hebbian synaptic modification rule was first introduced in [5]. In Hebbian learning rule, the biological synaptic weights change in proportion to the

correlation between the presynaptic and postsynaptic signals. For a single neuron, the Hebbian rule can be written as

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta y(t)\boldsymbol{x}(t), \tag{3.1}$$

where the learning rate $\eta > 0$, $\boldsymbol{w} \in \Re^n$ is the weight vector, $\boldsymbol{x}(t) \in \Re^n$ is an input vector at time t, $y(t)$ is the output of the neuron defined by $y(t) = \boldsymbol{w}^{\mathrm{T}}(t)\boldsymbol{x}(t)$.

The convergence of Hebbian rule can be briefly analyzed as follows.

For a stochastic input vector $\boldsymbol{x}$, assuming that $\boldsymbol{x}$ and $\boldsymbol{w}$ are uncorrelated, the expected weight change is given by

$$\mathrm{E}[\Delta \boldsymbol{w}] = \eta \mathrm{E}[y\boldsymbol{x}] = \eta \mathrm{E}\left[\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}\right] = \eta \boldsymbol{C}\mathrm{E}[\boldsymbol{w}], \tag{3.2}$$

where $\mathrm{E}[\cdot]$ is the expectation operator, and $C = \mathrm{E}[\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}]$ is the autocorrelation matrix of $\boldsymbol{x}$.

At equilibrium, $\mathrm{E}[\Delta \boldsymbol{w}] = 0$, and hence, it holds that the deterministic equation $\boldsymbol{C}\boldsymbol{w} = 0$. Due to the effect of noise terms, C is a full-rank positive-definite Hermitian matrix with positive eigenvalues $\lambda_i, i = 1, 2, \ldots, n$, and the associated orthogonal eigenvectors $\boldsymbol{c}_i$, where $n = \mathrm{rank}(C)$. Thus, $\boldsymbol{w} = 0$ is the only equilibrium state.

Equation (3.1) can be further represented in the continuous-time form

$$\dot{\boldsymbol{w}} = y\boldsymbol{x}. \tag{3.3}$$

Taking expectation on both sides, it holds that

$$\mathrm{E}[\dot{\boldsymbol{w}}] = \mathrm{E}[y\boldsymbol{x}] = \mathrm{E}\left[\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}\right] = \boldsymbol{C}\mathrm{E}[\boldsymbol{w}]. \tag{3.4}$$

This can be derived by minimizing the average instantaneous criterion function [6]

$$\mathrm{E}[E_{\mathrm{Hebb}}] = -\frac{1}{2}\mathrm{E}\left[y^2\right] = -\frac{1}{2}\mathrm{E}\left[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}\right] = -\frac{1}{2}\mathrm{E}\left[\boldsymbol{w}^{\mathrm{T}}\right]\boldsymbol{C}\mathrm{E}[\boldsymbol{w}], \tag{3.5}$$

where E_{Hebb} is the instantaneous criterion function. At equilibrium, $\mathrm{E}\left[\frac{\partial E_{\mathrm{Hebb}}}{\partial \boldsymbol{w}}\right] = -\boldsymbol{C}\mathrm{E}[\boldsymbol{w}] = \boldsymbol{0}$, thus $\boldsymbol{w} = \boldsymbol{0}$. Since $\mathrm{E}[\mathrm{H}(\boldsymbol{w})] = \mathrm{E}\left[\frac{\partial E^2_{\mathrm{Hebb}}}{\partial^2 \boldsymbol{w}}\right] = -\boldsymbol{C}$ is nonpositive for all $\mathrm{E}[\boldsymbol{w}]$, the solution $\boldsymbol{w} = \boldsymbol{0}$ is unstable, which drives $\boldsymbol{w}$ to infinite magnitude, with a direction parallel to that of the eigenvector of $\boldsymbol{C}$ associated with the largest eigenvalue [6]. Thus, the Hebbian rule is divergent.

To prevent the divergence of the Hebbian rule, one can normalize $\|\boldsymbol{w}\|$ to unity after each iteration [7]. This leads to the normalized Hebbian rule. Several other methods such as Oja's rule [8], Yuille's rule [9], Linsker's rule [10, 11], and Hassoun's rule [12] add a weight-decay term to the Hebbian rile to stabilize the algorithm.

3.3.2 Oja's Learning Rule

By adding a weight decay term into the Hebbian rule, Oja's learning rule was proposed in [8] and given by

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta y(t)\boldsymbol{x}(t) - \eta y^2(t)\boldsymbol{w}(t). \tag{3.6}$$

Oja's rule converges to a state that minimizes (3.5) subject to $\|\boldsymbol{w}\| = 1$. The solution is the principal eigenvector of $\mathbf{C}$. For small η, Oja's rule is proved to be equivalent to the normalized Hebbian rule [8].

Using the stochastic learning theory, the continuous-time version of Oja's rule is given by a nonlinear stochastic differential equation

$$\dot{\boldsymbol{w}} = \eta(y\boldsymbol{x} - y^2\boldsymbol{w}). \tag{3.7}$$

The corresponding deterministic equation based on statistical average is thus derived as

$$\dot{\boldsymbol{w}} = \eta\left[\boldsymbol{C}\boldsymbol{w} - \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{w}\right)\boldsymbol{w}\right]. \tag{3.8}$$

At equilibrium, it holds that

$$\boldsymbol{C}\boldsymbol{w} = \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{w}\right)\boldsymbol{w}. \tag{3.9}$$

It can be easily seen that the solutions are $\boldsymbol{w} = \pm\boldsymbol{c}_i, i = 1, 2, \ldots, n$, whose associated eigenvalues λ_i are arranged in a descending order as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$.

Note that the average Hessian

$$\mathrm{H}(\boldsymbol{w}) = \frac{\partial}{\partial \boldsymbol{w}}\left[-\boldsymbol{C}\boldsymbol{w} + \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{w}\right)\boldsymbol{w}\right] = -\boldsymbol{C} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{C}\boldsymbol{w}\boldsymbol{I} + 2\boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{C} \tag{3.10}$$

is positive-definite only at $\boldsymbol{w} = \pm\boldsymbol{c}_1$, if $\lambda_1 \neq \lambda_2$ [12], where I is an $n \times n$ identity matrix. This can be seen from

$$\begin{aligned}\mathrm{H}(\boldsymbol{c}_i)\boldsymbol{c}_j &= (\lambda_i - \lambda_j)\boldsymbol{c}_j + 2\lambda_j\boldsymbol{c}_i\boldsymbol{c}_i^{\mathrm{T}}\boldsymbol{c}_j \\ &= \begin{cases} 2\lambda_i 2\boldsymbol{c}_i & i = j \\ (\lambda_i - \lambda_j)\boldsymbol{c}_j & i \neq j \end{cases}.\end{aligned} \tag{3.11}$$

Thus, Oja's rule always converges to the principal component of $\boldsymbol{C}$.

The convergence analysis of the stochastic discrete-time algorithms such as the gradient descent method is conventionally based on the stochastic approximation theory [13]. A stochastic discrete-time algorithm is first converted into deterministic continuous-time ODEs, and then analyzed by using Lyapunov's second theorem.

This conversion is based on the Robbins–Monro conditions, which require the learning rate to gradually approach zero as $t \to \infty$. This limitation is not practical for implementation, especially for the learning of nonstationary data. In [14], Zufiria proposed to convert the stochastic discrete-time algorithms into their deterministic discrete-time formulations that characterize their average evolution from a conditional expectation perspective. This method has been applied to Oja's rule and the dynamics have been analyzed, and chaotic behavior has been observed in some invariant subspaces. Such analysis can guarantee the convergence of the Oja's rule by selecting some constant learning rate. A constant learning rate for fast convergence has also been suggested as $\eta = 0.618\ \lambda_1$ [15]. Recently, the convergence of many PCA algorithms of Oja's rule type have been analyzed by using deterministic discrete-time methods, the details of which will be discussed in Chap. 6.

3.4 Hebbian/Anti-Hebbian Rule-Based Principal Component Analysis

Hebbian rule-based PCA algorithms include the single PCA algorithm, multiple PCA algorithms and principal subspace analysis algorithm. These neural PCA algorithms originate from the seminal work by Oja [8]. The output of the neuron is updated by $\boldsymbol{y} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$, where $\boldsymbol{w} = (w_1, w_2, \ldots, w_{J1})^{\mathrm{T}}$. Here the activation function is the linear function $\varphi(x) = x$. The PCA turns out to be closely related to the Hebbian rule.

The PCA algorithms discussed in this section are based on the Hebbian rule. The network model was first proposed by Oja [16], where a J_1–J_2 FNN is used to extract the first J_2 PCs. The architecture of the PCA network is shown in Fig. 3.1, which is

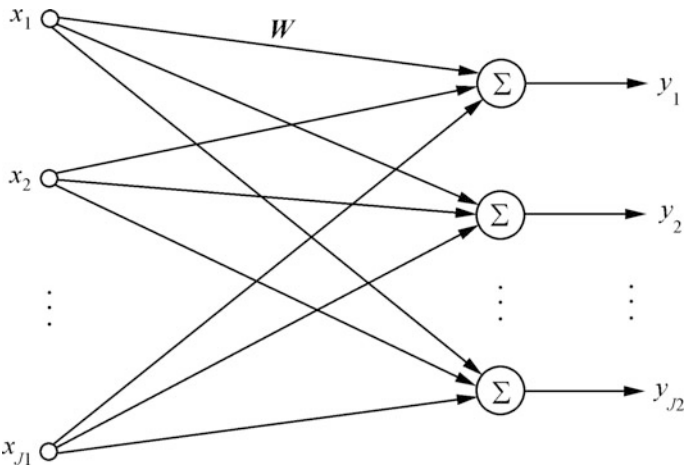

Fig. 3.1 Architecture of the PCA network

a simple expansion of the single-neuron PCA model. The output of the network is given by $\boldsymbol{y} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}$, where $\boldsymbol{y} = (y_1, y_2, \ldots, y_{J2})^{\mathrm{T}}$, $\boldsymbol{x} = (x_1, x_2, \ldots, x_{J1})^{\mathrm{T}}$, $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_{J2}]$, $\boldsymbol{w}_i = (w_{1i}, w_{2i}, \ldots, w_{J1i})^{\mathrm{T}}$.

3.4.1 Subspace Learning Algorithms

By using Oja's learning rule, $\boldsymbol{w}$ will converge to a unit eigenvector of the correlation matrix $\boldsymbol{C}$, and the variance of the output y is maximized. For zero-mean input data, this extracts the first PC. Here Oja's learning rule can be rewritten for the convenience of presentation as

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta y(t)\boldsymbol{x}(t) - \eta y^2(t)\boldsymbol{w}(t), \tag{3.12}$$

where the term $y(t)\boldsymbol{x}(t)$ is the Hebbian term, and $-y^2(t)\boldsymbol{w}(t)$ is a decaying term, which is used to prevent instability. In order to keep the algorithm convergent, it is proved that $0 < \eta(t) < 1/1.2\lambda_1$ is required [16], where λ_1 is the largest eigenvalue of $\mathbf{C}$. If $\eta(t) \geq 1/\lambda_1$, $\mathbf{w}$ will not converge to $\pm\mathbf{c}1$ even if it is initially close to the target [17].

3.4.1.1 Symmetrical Subspace Learning Algorithm

Oja proposed a learning algorithm for the PCA network, referred to as the symmetrical subspace learning algorithm (SLA) [16]. The SLA can be derived by maximizing

$$E_{\mathrm{SLA}} = \frac{1}{2}\mathrm{tr}\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{W}\right) \quad \text{subject to} \quad \boldsymbol{W}^{\mathrm{T}}\boldsymbol{W} = \boldsymbol{I}, \tag{3.13}$$

where $\boldsymbol{I}$ is a $\mathrm{J}_2 \times \mathrm{J}_2$ identity matrix. The SLA is given as [16]

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)y_i(t)[\boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)], \tag{3.14}$$

$$\hat{\boldsymbol{x}}(t) = \boldsymbol{W}\boldsymbol{y}. \tag{3.15}$$

After the algorithm converges, $\boldsymbol{W}$ is roughly orthonormal and the columns of $\boldsymbol{W}$, namely $\boldsymbol{w}_i, i = 1, 2, \ldots, J_2$, converge to some linear combination of the first J_2 principal eigenvectors of $\boldsymbol{C}$ [16], which is a rotated basis of the dominant eigenvector subspace. The value of $\boldsymbol{w}_i$ is dependent on the initial condition and the training samples.

The corresponding eigenvalues $\lambda_i, i = 1, 2, \ldots, J_2$, which approximate $E\left[y_i^2\right]$, can be adaptively estimated by

$$\hat{\lambda}_i(t+1) = \left(1 - \frac{1}{t+1}\right)\hat{\lambda}_i(t) + \frac{1}{t+1}y_i^2(t+1). \tag{3.16}$$

The PCA performs optimally when there is no noise process involved.

3.4.1.2 Weighted Subspace Learning Algorithm

The weighted SLA can be derived by maximizing the same criterion (3.13), with the constraint changed to $\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W} = \boldsymbol{\alpha}$, where $\boldsymbol{\alpha} = diag(\alpha_1, \alpha_2, \ldots, \alpha_{J_2})$, is an arbitrary diagonal matrix with $\alpha_1 > \alpha_2 > \cdots > \alpha_{J_2} > 0$.

The weighted SLA is given by [18, 19]

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)y_i(t)[\boldsymbol{x}(t) - \gamma_i\hat{\boldsymbol{x}}(t)], \tag{3.17}$$

$$\hat{\boldsymbol{x}}(t) = \boldsymbol{W}\boldsymbol{y}, \tag{3.18}$$

for $i = 1, 2, \ldots, J_2$, where $\gamma_i, i = 1, 2, \ldots, J_2$, are coefficients satisfying $0 < \gamma_1 < \gamma_2 < \cdots < \gamma_{J_2}$.

Due to the asymmetry introduced by γ_i, $\boldsymbol{w}_i$ almost surely converges to the eigenvectors of $\boldsymbol{C}$. The weighted subspace algorithm can perform the PCA, however, norms of the weight vectors are not equal to unity.

The subspace and weighted subspace algorithms are nonlocal algorithms relying on the calculation of the errors and the backward propagation of the values between the layers [3]. Several algorithms converting PSA into PCA have been proposed, the details can be found in [3].

3.4.2 Generalized Hebbian Algorithm

By combining Oja's rule and the GSO procedure, Sanger proposed the GHA for extracting the first J_2 PCs [20]. The GHA can extract the first J_2 eigenvectors in the order of decreasing eigenvalues.

The GHA is given by [20]

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta_i(t)y_i(t)[\boldsymbol{x}(t) - \hat{\boldsymbol{x}}_i(t)], \tag{3.19}$$

$$\hat{\boldsymbol{x}}_i(t) = \sum\nolimits_{j=1}^{i} \boldsymbol{w}_j(t)y_j(t), \tag{3.20}$$

for $i = 1, 2, \ldots, J_2$. The GHA becomes a local algorithm by solving the summation term in (3.20) in a recursive form

$$\hat{\boldsymbol{x}}_i(t) = \hat{\boldsymbol{x}}_{i-1}(t) + \boldsymbol{w}_i(t) y_i(t), \tag{3.21}$$

for $i = 1, 2, \ldots, J_2$, where $\hat{\boldsymbol{x}}_0(t) = 0$. $\eta_i(t)$ is usually selected the same for all neurons. When $\eta_i = \eta$ for all i, the algorithm can be written in a matrix form

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) - \eta \boldsymbol{W}(t) \mathrm{LT}\left[\boldsymbol{y}(t)\boldsymbol{y}^T(t)\right] + \eta \boldsymbol{x}(t)\boldsymbol{y}^T(t), \tag{3.22}$$

where the operator LT[·] selects the lower triangle of input matrix. In the GHA, the mth neuron converges to the mth PC, and all the neurons tend to converge together. $\boldsymbol{w}_i$ and $E\left[y_i^2\right]$ approach $\boldsymbol{c}_i$ and λ_i, respectively, as $t \to \infty$.

Both the SLA and GHA algorithms employ implicit or explicit GSO to decorrelate the connection weights from one another. The weighted SLA algorithm performs well for extracting less-dominant components.

3.4.3 Learning Machine for Adaptive Feature Extraction via PCA

Learning machine for adaptive feature extraction via principal component analysis is called LEAP algorithm, and it is another local PCA algorithm for extracting all the J_2 PCs and their corresponding eigenvectors. The LEAP is given by

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta\{\boldsymbol{B}_i(t) y_i(t)[\boldsymbol{x}(t) - \boldsymbol{w}_i(t) y_i(t)] - \boldsymbol{A}_i(t)\boldsymbol{w}_i(t)\}, \tag{3.23}$$

for $i = 1, 2, \ldots, J_2$, where η is the learning rate, $y_i(t)\boldsymbol{x}(t)$ is a Hebbian term, and

$$\boldsymbol{A}_i(t) = \begin{cases} 0, & i = 1 \\ \boldsymbol{A}_{i-1}(t) + \boldsymbol{w}_{i-1}(t)\boldsymbol{w}_{i-1}^T(t), & i = 2, \ldots, J_2 \end{cases}, \tag{3.24}$$

$$\boldsymbol{B}_i(t) = \boldsymbol{I} - \boldsymbol{A}_i(t), \quad i = 1, 2, \cdots, J_2. \tag{3.25}$$

The $J_1 \times J_1$ matrices $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ are important decorrelating terms for performing the GSO among all weights at each iteration. Unlike the SLA [16] and GHA [20] algorithms, whose stability analyses are based on the stochastic approximation theory [13], the stability analysis of the LEAP algorithm is based on Lyapunov's first theorem, and η can be selected as a small positive constant. Due to the use of a constant learning rate, the LEAP is capable of tracking nonstationary processes. The LEAP can satisfactorily extract PCs even for ill-conditioned autocorrelation matrices.

3.4.4 The Dot-Product-Decorrelation Algorithm (DPD)

The DPD algorithm is a nonlocal PCA algorithm, and it moves $\boldsymbol{w}_i, i = 1, 2, \ldots, J_2$, toward the J_2 principal eigenvectors c_i, ordered arbitrarily

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)\left[\boldsymbol{x}(t)y_i(t) - \left(\sum_{j=1}^{J_2} \boldsymbol{w}_j(t)\boldsymbol{w}_j^{\mathrm{T}}(t)\right)\frac{\boldsymbol{w}_i(t)}{\|\boldsymbol{w}_i(t)\|}\right], \tag{3.26}$$

where $\eta(t)$ satisfies the Robbins–Monro conditions. The algorithm induces the norms of the weight vectors toward the corresponding eigenvalues, i.e., $\|\boldsymbol{w}_i(t)\| \rightarrow \lambda_i(t)$, as $t \rightarrow \infty$. The algorithm is as fast as the GHA [20], weighted SLA [18, 19], and least mean squared error reconstruction (LMSER) [21] algorithms.

3.4.5 Anti-Hebbian Rule-Based Principal Component Analysis

When the update of a synaptic weight is proportional to the correlation of the presynaptic and postsynaptic activities, and the direction of the change is opposite to that in the Hebbian rule, the learning rule is called an anti-Hebbian learning rule [3]. The anti- Hebbian rule can be used to remove correlations between units receiving correlated inputs [22, 23], and it is inherently stable.

Anti-Hebbian rule-based PCA algorithms can be derived by using a network architecture of the J_1–J_2 FNN with lateral connections among the output units [22, 23]. The lateral connections can be in a symmetrical or hierarchical topology. A hierarchical lateral connection topology is illustrated in Fig. 3.2, based on which the Rubner–Tavan PCA algorithm [22, 23] and the APEX [24] were proposed. In [25], the local PCA algorithm is based on a full lateral connection topology. The feedforward weight matrix $\boldsymbol{W}$ is described in the preceding sections, and the

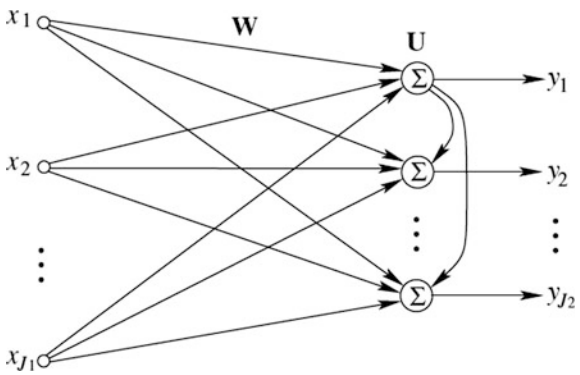

Fig. 3.2 Architecture of the PCA network with hierarchical lateral connections. The lateral weight matrix $\boldsymbol{U}$ is an upper triangular matrix with the diagonal elements being zero

lateral weight matrix $\boldsymbol{U} = [\boldsymbol{u}_1 \ \ldots \ \boldsymbol{u}_{J2}]$ is a $J_2 \times J_2$ matrix, where $\boldsymbol{u}_i = (u_{1i}, u_{2i}, \ldots, u_{J2i})^{\mathrm{T}}$ includes all the lateral weights connected to neuron i and u_{ji} denotes the lateral weight from neuron j to neuron i.

3.4.5.1 Rubner-Tavan PCA Algorithm

The Rubner-Tavan PCA algorithm is based on the PCA network with hierarchical lateral connection topology [22, 23]. The algorithm extracts the first J_2 PCs in a decreasing order of the eigenvalues. The output of the network is given by [22, 23]

$$y_i = \boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{u}_i^{\mathrm{T}}\boldsymbol{y}, \quad i = 1, 2, \ldots, J_2. \tag{3.27}$$

Note that $u_{ji} = 0$ for $j \geq i$ and $\boldsymbol{U}$ is a $J_2 \times J_2$ upper triangular matrix.

The weights $\boldsymbol{w}_i$ are trained by Oja's rule, and the lateral weights $\boldsymbol{u}_i$ are updated by the anti-Hebbian rule

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta_1(t) y_i(t)[\boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)], \tag{3.28}$$

$$\hat{\boldsymbol{x}} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{y}, \tag{3.29}$$

$$\boldsymbol{u}_i(t+1) = \boldsymbol{u}_i(t) - \eta_2 y_i(t)\boldsymbol{y}(t). \tag{3.30}$$

This is a nonlocal algorithm. Typically, the learning rate $\eta_1 = \eta_2 > 0$ is selected as a small number between 0.001 and 0.1 or according to a heuristic derived from the Robbins–Monro conditions. During the training process, the outputs of the neurons are gradually uncorrelated and the lateral weights approach zero. The network should be trained until the lateral weights $\boldsymbol{u}_i$ are below a specified level.

3.4.5.2 APEX Algorithm

The APEX algorithm is used to adaptively extract the PCs [24]. The algorithm is recursive and adaptive, namely, given $i - 1$ PCs, it can produce the ith PC iteratively. The hierarchical structure of lateral connections among the output units serves the purpose of weight orthogonalization. This structure also allows the network to grow or shrink without retraining the old units. The convergence analysis of the APEX algorithm is based on the stochastic approximation theory, and the APEX is proved to have the property of exponential convergence.

Assuming that the correlation matrix $\boldsymbol{C}$ has distinct eigenvalues arranged in the decreasing order as $\lambda_1 > \lambda_2 > \cdots > \lambda_{J2}$ with the associated eigenvectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{J2}$, the algorithm is given by [24, 26]

$$\boldsymbol{y} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}, \tag{3.31}$$

$$y_i = \boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{u}^{\mathrm{T}}\boldsymbol{y}, \tag{3.32}$$

where $\boldsymbol{y} = (y_1, \ldots, y_{i-1})^{\mathrm{T}}$ is the output vector, $\boldsymbol{u} = (u_{1i}, u_{2i}, \ldots, u_{(i-1)i})^{\mathrm{T}}$, and $\boldsymbol{W} = [\boldsymbol{w}_1 \ldots \boldsymbol{w}_{i-1}]$ is the weight matrix of the first $i - 1$ neurons. These definitions are for the first i neurons, which are different from their respective definitions given in the preceding sections. The iteration is given as [24, 26]

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta_i(t)\left[y_i(t)\boldsymbol{x}(t) - y_i^2(t)\boldsymbol{w}_i(t)\right], \tag{3.33}$$

$$\boldsymbol{u}(t+1) = \boldsymbol{u}(t) - \eta_i(t)\left[y_i(t)\boldsymbol{y}(t) + y_i^2(t)\boldsymbol{u}(t)\right]. \tag{3.34}$$

Equations (3.33) and (3.34) are respectively the Hebbian and anti-Hebbian parts of the algorithm. y_i tends to be orthogonal to all the previous components due to the anti-Hebbian rule, also called the orthogonalization rule.

Both sequential and parallel APEX algorithms have been presented in [26]. In the parallel APEX, all J_2 output neurons work simultaneously. In the sequential APEX, the output neurons are added one by one. The sequential APEX is more attractive in practical applications, since one can decide a desirable number of neurons during the learning process. The APEX algorithm is especially useful when the number of required PCs is not known a priori. When the environment is changing over time, a new PC can be added to compensate for the change without affecting the previously computed principal components. Thus, the network structure can be expanded if necessary.

The stopping criterion can be that for each i the changes in $\boldsymbol{w}_i$ and $\boldsymbol{u}$ are below a threshold. At this time, $\boldsymbol{w}_i$ converges to the eigenvector of the correlation matrix $\boldsymbol{C}$ associated with the ith largest eigenvalue, and $\boldsymbol{u}$ converges to zero. The stopping criterion can also be that the change of the average output variance $\sigma_i^2(t)$ is sufficiently small.

Most existing linear complexity methods including the GHA [20], the SLA [16], and the PCA with the lateral connections require a computational complexity of $O(J_1J_2)$ per iteration. For the recursive computation of each additional PC, the APEX requires $O(J_1)$ operations per iteration, while the GHA utilizes $O(J_1J_2)$ per iteration. In contrast to the heuristic derivation of the APEX, a class of learning algorithms, called the *ψ-APEX*, is presented based on criterion optimization [27]. ψ can be selected as any function that guarantees the stability of the network. Some members in the class have better numerical performance and require less computational effort compared to that of both the GHA and the APEX.

3.5 Least Mean Squared Error-Based Principal Component Analysis

Existing PCA algorithms including the Hebbian rule-based algorithms can be derived by optimizing an objective function using the gradient descent method. The least mean squared error (LMSE)-based methods are derived from the modified MSE function

$$E(\boldsymbol{W}) = \sum_{t_1=1}^{t} \mu^{t-t_1} \left\| \boldsymbol{x}_{t_1} - \boldsymbol{W}\boldsymbol{W}^T \boldsymbol{x}_{t_1} \right\|^2, \tag{3.35}$$

where $0 < \mu \leq 1$ is a forgetting factor used for nonstationary observation sequences, and t is the current time instant. Many adaptive PCA algorithms actually optimize (3.35) by using the gradient descent method [21, 28] and the RLS method [28–32].

The gradient descent or Hebbian rule-based algorithms are highly sensitive to parameters such as η. It is difficult to choose proper parameters guaranteeing both a small misadjustment and a fast convergence. To overcome these drawbacks, applying the RLS to the minimization of (3.35) yields the RLS-based algorithms such as the adaptive principal components extraction (APEX) [24, 26], the Kalman-type RLS [29], the projection approximation subspace tracking (PAST) [28], the PAST with deflation (PASTd) [28], and the robust RLS algorithm (RRLSA) [31].

All RLS-based PCA algorithms exhibit fast convergence and high tracking accuracy and are suitable for slow changing nonstationary vector stochastic processes. All these algorithms correspond to a three-layer J_1-J_2-J_1 linear autoassociative network model, and they can extract all the J_2 PCs in a descending order of the eigenvalues, where a GSO-like orthonormalization procedure is used.

3.5.1 *Least Mean Square Error Reconstruction Algorithm (LMSER)*

The LMSER algorithm was derived based on the MSE criterion using the gradient descent method [21]. The LMSER algorithm can be written as

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)\{2\boldsymbol{A}(t) - \boldsymbol{C}_i(t)\boldsymbol{A}(t) - \boldsymbol{A}(t)\boldsymbol{C}_i(t) - \gamma[\boldsymbol{B}_i(t)\boldsymbol{A}(t) + \boldsymbol{A}(t)\boldsymbol{B}_i(t)]\}\boldsymbol{w}_i(t), \tag{3.36}$$

for $i = 1, 2, \ldots, J_2$, where $\boldsymbol{A}(t) = \boldsymbol{x}(t)\boldsymbol{x}^{\mathrm{T}}(t)$, $\boldsymbol{C}_i(t) = \boldsymbol{w}_i(t)\boldsymbol{w}_i^{\mathrm{T}}(t), i = 1, 2, \ldots, J_2$, $\boldsymbol{B}_i(t) = \boldsymbol{B}_{i-1}(t) + \boldsymbol{C}_{i-1}(t)$, $i = 2, \ldots, J_2$, and $\boldsymbol{B}_1(t) = \boldsymbol{0}$. The selection of $\eta(t)$ is based on the Robbins–Monro conditions and $\gamma \geq 1$.

The LMSER reduces to Oja's algorithm when $\boldsymbol{W}(t)$ is orthonormal, namely $\boldsymbol{W}^{\mathrm{T}}(t)\boldsymbol{W}(t) = \boldsymbol{I}$. Because of this, Oja's algorithm can be treated as an approximate stochastic gradient rule to minimize the MSE. Increasing the values of γ and δ results in a larger asymptotic MSE but faster convergence and vice versa, namely the stability speed problem. The LMSER uses nearly twice as much computation as the weighted SLA [18, 19] and the GHA [20], for each update of the weight. However, it leads to a smaller asymptotic and faster convergence for the minor eigenvectors [33].

3.5.2 Projection Approximation Subspace Tracking Algorithm (PAST)

The PASTd [28] is a well-known subspace tracking algorithm updating the signal eigenvectors and eigenvalues. The PASTd is based on the PAST. Both the PAST and the PASTd are derived for complex-valued signals, which are very common in signal processing area. At iteration t, the PASTd algorithm is given as [28]

$$y_i(t) = \boldsymbol{w}_i^H(t-1)\boldsymbol{x}_i(t), \tag{3.37}$$

$$\delta_i(t) = \mu\delta_i(t-1) + |y_i(t)|^2, \tag{3.38}$$

$$\hat{\boldsymbol{x}}_i(t) = \boldsymbol{w}_i(t-1)y_i(t), \tag{3.39}$$

$$\boldsymbol{w}_i(t) = \boldsymbol{w}_i(t-1) + [\boldsymbol{x}_i(t) - \hat{\boldsymbol{x}}_i(t)]\frac{y_i^*(t)}{\delta_i(t)}, \tag{3.40}$$

$$\boldsymbol{x}_{i+1}(t) = \boldsymbol{x}_i(t) - \boldsymbol{w}_i(t)y_i(t), \tag{3.41}$$

for $i = 1, \ldots, J_2$, where $\boldsymbol{x}_1(t) = \boldsymbol{x}_t$, and the superscript * denotes the conjugate operator.

$\boldsymbol{w}_i(0)$ and $\delta_i(0)$ should be suitably selected. $\boldsymbol{W}(0)$ should contain J_2 orthonormal vectors, which can be calculated from an initial block of data or from arbitrary initial data. A simple way is to set $\boldsymbol{W}(0)$ as the J_2 leading unit vectors of the $J_1 \times J_1$ identity matrix. $\delta_i(0)$ can be set as unity. The choice of these initial values affects the transient behavior, but not the steady-state performance of the algorithm. $\boldsymbol{w}_i(t)$ provides an estimate of the ith eigenvector, and $\delta_i(t)$ is an exponentially weighted estimate of the associated eigenvalue.

Both the PAST and the PASTd have linear computational complexity, that is, $O(J_1J_2)$ operations in every update, as in the cases of the SLA [16], the GHA [20], the LMSER [21], and the novel information criterion (NIC) algorithm [30]. The PAST computes an arbitrary basis of the signal subspace, while the PASTd is able to update the signal eigenvectors and eigenvalues. Both algorithms produce nearly orthonormal, but not exactly orthonormal, subspace basis or eigenvector estimates. If perfectly orthonormal eigenvector estimates are required, an orthonormalization procedure is necessary. The Kalman-type RLS [29] combines the basic RLS algorithm with the GSO procedure in a manner similar to that of the GHA. The Kalman-type RLS and the PASTd are exactly identical if the inverse of the covariance of the output of the ith neuron, $P_i(t)$, in the Kalman-type RLSA is set as $1/\delta_i(t)$ in the PASTd.

In the one-unit case, both the PAST and PASTd are identical to Oja's learning rule except that the PAST and the PASTd have a self-tuning learning rate $1/\delta_1(t)$. Both the PAST and the PASTd provide much more robust estimates than the EVD

and converge much faster than the SLA [16]. The PASTd has been extended for the tracking of both the rank and the subspace by using information theoretic criteria such as the AIC and the MDL [34].

3.5.3 Robust RLS Algorithm (RRLSA)

The RRLSA [31] is more robust than the PASTd [28]. The RRLSA can be implemented in a sequential or parallel manner. Given the ith neuron, the sequential algorithm is given for all patterns as [31]

$$\overline{\overline{\boldsymbol{w}}}_i(t-1) = \frac{\boldsymbol{w}_i(t-1)}{\|\boldsymbol{w}_i(t-1)\|}, \tag{3.42}$$

$$y_i(t) = \overline{\overline{\boldsymbol{w}}}_i^{\mathrm{T}}(t-1)\boldsymbol{x}(t), \tag{3.43}$$

$$\hat{\boldsymbol{x}}_i(t) = \sum_{j=1}^{i-1} y_j(t)\overline{\overline{\boldsymbol{w}}}_j(t-1), \tag{3.44}$$

$$\boldsymbol{w}_i(t) = \mu\boldsymbol{w}_i(t-1) + [\boldsymbol{x}_i(t) - \hat{\boldsymbol{x}}_i(t)]y_i(t), \tag{3.45}$$

$$\hat{\lambda}_i(t) = \frac{\|\boldsymbol{w}_i(t)\|}{t}, \tag{3.46}$$

for $i = 1, \ldots, J_2$, where y_i is the output of the ith hidden unit, and $\boldsymbol{w}_i(0)$ is initialized as a small random value. By changing (3.44) into a recursive form, the RRLSA becomes a local algorithm.

The RRLSA has the same flexibility as the Kalman-type RLS [29], the PASTd, and the APEX, in that increasing the number of neurons does not affect the previously extracted principal components. The RRLSA naturally selects the inverse of the output energy as the adaptive learning rate for the Hebbian rule. The Hebbian and Oja rules are closely related to the RRLSA algorithm by suitable selection of the learning rates [31].

The RRLSA is also robust to the error accumulation from the previous components, which exists in the sequential PCA algorithms such as the Kalman-type RLS and the PASTd. The RRLSA converges rapidly, even if the eigenvalues extend over several orders of magnitude. According to the empirical results [31], the RRLSA provides the best performance in terms of convergence speed as well as steady-state error, whereas the Kalman-type RLS and the PASTd have similar performance, which is inferior to that of the RRLSA.

3.6 Optimization-Based Principal Component Analysis

The PCA can be derived by many optimization methods based on a properly defined objective function. This leads to many other algorithms, including gradient descent-based algorithms [9–11, 35], the CG method [36], and the quasi-Newton method [37, 38]. The gradient descent method usually converges to a local minimum. Second-order algorithms such as the CG and quasi-Newton methods typically converge much faster than first-order methods but have a computational complexity of $O(J_1^2 J_2)$ per iteration.

The infomax principle [10, 11] was first proposed by Linsker to describe a neural network algorithm. The principal subspace is derived by maximizing the mutual information criterion. Other examples of information criterion-based algorithms are the NIC algorithm [30] and the coupled PCA [39].

3.6.1 Novel Information Criterion (NIC) Algorithm

The NIC algorithm [30] is obtained by applying the gradient descent method to maximize the NIC. The NIC is a cost function very similar to the mutual information criterion [10, 11] but integrates a soft constraint on the weight orthogonalization

$$E_{\mathrm{NIC}} = \frac{1}{2}\left\{\ln\left(\det(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{W})\right) - \mathrm{tr}(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W})\right\}. \tag{3.47}$$

Unlike the MSE, the NIC has a steep landscape along the trajectory from a small weight matrix to the optimum one. E_{NIC} has a single global maximum, and all the other stationary points are unstable saddle points. At the global maximum

$$E_{\mathrm{NIC}}^{*} = \frac{1}{2}\left(\sum_{i=1}^{J_2} \ln\, \lambda_i - J_2\right), \tag{3.48}$$

$\boldsymbol{W}$ yields an arbitrary orthonormal basis of the principal subspace.

The NIC algorithm was derived from E_{NIC} by using the gradient descent method, and the algorithm is given as

$$\boldsymbol{W}(t+1) = (1-\eta)\boldsymbol{W}(t) + \eta\hat{\boldsymbol{C}}(t+1)\boldsymbol{W}(t)\left[\boldsymbol{W}^{\mathrm{T}}(t)\hat{\boldsymbol{C}}(t+1)\boldsymbol{W}(t)\right]^{-1}, \tag{3.49}$$

where $\hat{\boldsymbol{C}}(t)$ is the estimate of the covariance matrix $\boldsymbol{C}(t)$

$$\hat{\boldsymbol{C}}(t) = \frac{1}{t}\sum_{i=1}^{t} \mu^{t-i}\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}} = \mu\frac{t-1}{t}\hat{\boldsymbol{C}}(t-1) + \frac{1}{t}\boldsymbol{x}_i\boldsymbol{x}_i^{\mathrm{T}} \tag{3.50}$$

and $\mu \in (0, 1]$ is a forgetting factor. The NIC algorithm has a computational complexity of $O(J_1^2 J_2)$ per iteration.

Like the PAST algorithm [28], the NIC algorithm is a PSA method. It can extract the principal eigenvectors when the deflation technique is incorporated. The NIC algorithm converges much faster than the SLA and the LMSER and can globally converge to the PSA solution from almost any weight initialization. Reorthormalization can be applied so as to perform true PCA [30].

By selecting a well-defined adaptive learning rate, the NIC algorithm can also generalize some well-known PSA/PCA algorithms. For online implementation, an RLS version of the NIC algorithm has also been given in [30]. The PAST algorithm [28] is a special case of the NIC algorithm when η is unity, and the NIC algorithm essentially represents a robust improvement of the PAST.

In order to break the symmetry in the NIC, the weighted information criterion (WINC) [32] was proposed by adding a weight to the NIC. Two WINC algorithms are, respectively, derived by using the gradient ascent and the RLS. The gradient ascent-based WINC algorithm can be viewed as an extended weighted SLA with an adaptive step size, leading to a much faster convergence speed. The RLS-based WINC algorithm has not only fast convergence and high accuracy, but also a low computational complexity.

3.6.2 Coupled Principal Component Analysis

The most popular PCA or MCA algorithms do not consider eigenvalue estimates in the update of the weights, and they suffer from the stability speed problem because the eigen motion depends on the eigenvalues of the covariance matrix [39]. The convergence speed of a system depends on the eigenvalues of its Jacobian. In PCA algorithms, the eigen motion depends on the principal eigenvalue of the covariance matrix, while in MCA algorithms it depends on all eigenvalues [39].

Coupled learning rules can be derived by applying the Newton method to a common information criterion. In coupled PCA/MCA algorithms, both the eigenvalues and eigenvectors are simultaneously adapted. The Newton method yields averaged systems with identical speed of convergence in all eigen directions. The Newton descent-based PCA and MCA algorithms, respectively called nPCA and nMCA, are derived by using the information criterion [39]:

$$E_{\text{coupled}}(\boldsymbol{w}, \lambda) = \frac{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{C} \boldsymbol{w}}{\lambda} - \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} + \ln \lambda, \tag{3.51}$$

where λ is an estimate of the eigenvalue.

By approximation $\boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} \approx 1$, the nPCA is reduced to the ALA [17]. Further approximating the ALA by $\boldsymbol{w}^{\mathrm{T}} \boldsymbol{C} \boldsymbol{w} \approx \lambda$ leads to an algorithm called cPCA. The cPCA is a stable PCA algorithm, but there may be fluctuation in the weight vector length in the iteration process. This problem can be avoided by explicitly

renormalizing the weight vector at every iteration, and this leads to the following robust PCA (rPCA) algorithm [39]:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta(t)\left(\frac{\boldsymbol{x}(t)y(t)}{\lambda(t)} - \boldsymbol{w}(t)\right), \tag{3.52}$$

$$\boldsymbol{w}(t+1) = \frac{\boldsymbol{w}(t+1)}{\|\boldsymbol{w}(t+1)\|}, \tag{3.53}$$

$$\lambda(t+1) = \lambda(t) + \eta(t)(y^2(t) - \lambda(t)), \tag{3.54}$$

where $\eta(t)$ is a small positive number and can be selected according to the Robbins–Monro conditions. The rPCA is shown to be closely related to the RRLSA algorithm [31] by applying the first-order Taylor approximation on the rPCA. The RRLSA can also be derived from the ALA algorithm by using the first-order Taylor approximation.

In order to extract multiple PCs, one has to apply an orthonormalization procedure, e.g., the GSO, or its first-order approximation as used in the SLA, or deflation as in the GHA. In the coupled learning rules, multiple PCs are simultaneously estimated by a coupled system of equations. It has been reported in [40] that in the coupled learning rules a first-order approximation of the GSO is superior to the standard deflation procedure in terms of orthonormality error and the quality of the eigenvectors and eigenvalues generated. An additional normalization step that enforces unit length of the eigenvectors further improves the orthonormality of the weight vectors [40].

3.7 Nonlinear Principal Component Analysis

The aforementioned PCA algorithms apply a linear transform to the input data. The PCA is based on the Gaussian assumption for data distribution, and the optimality of the PCA results from taking into account only the second-order statistics, namely the covariances. For non-Gaussian data distributions, the PCA is not able to capture complex nonlinear correlations, and nonlinear processing of the data is usually more efficient. Nonlinearities introduce higher-order statistics into the computation in an implicit way. Higher-order statistics, defined by cumulants or higher-than-second moments, are needed for a good characterization of non-Gaussian data.

The Gaussian distribution is only one of the canonical exponential distributions, and it is suitable for describing real-valued data. In the case of binary-valued, integer-valued, or non-negative data, the Gaussian assumption is inappropriate, and a family of exponential distributions can be used. For example, the Poisson distribution is better suited for integer data and the Bernoulli distribution to binary data, and an exponential distribution to nonnegative data. All these distributions

belong to the exponential family. The PCA can be generalized to distributions of the exponential family. This generalization is based on a generalized linear model and criterion functions using the Bregman distance. This approach permits hybrid dimensionality reduction in which different distributions are used for different attributes of the data.

When the feature space is nonlinearly related to the input space, we need to use nonlinear PCA. The outputs of nonlinear PCA networks are usually more independent than their respective linear cases. For non-Gaussian input data, the PCA may fail to provide an adequate representation, while a nonlinear PCA permits the extraction of higher-order components and provides a sufficient representation. Nonlinear PCA networks and learning algorithms can be classified into symmetric and hierarchical ones similar to those for the PCA networks. After training, the lateral connections between output units are not needed, and the network becomes purely feedforward. In the following, we discuss the kernel PCA, robust PCA, and nonlinear PCA.

3.7.1 Kernel Principal Component Analysis

Kernel PCA [41, 42] is a special, linear algebra-based nonlinear PCA, which introduces kernel functions into the PCA. The kernel PCA first maps the original input data into a high-dimensional feature space using the kernel method and then calculates the PCA in the high-dimensional feature space. The linear PCA in the high-dimensional feature space corresponds to a nonlinear PCA in the original input space.

Given an input pattern set $\{\boldsymbol{x}_i \in \Re^{J_1} | i = 1, 2, \ldots, N\}$, $\varphi : \Re^{J_1} \rightarrow \Re^{J_2}$ is a nonlinear map from the J_1-dimensional input to the J_2-dimensional feature space. A J_2-by-J_2 correlation matrix in the feature space is defined by

$$\boldsymbol{C}_1 = \frac{1}{N} \sum_{i=1}^{N} \varphi(\boldsymbol{x}_i) \varphi^{\mathrm{T}}(\boldsymbol{x}_i). \tag{3.55}$$

Like the PCA, the set of feature vectors is limited to zero mean

$$\frac{1}{N} \sum_{i=1}^{N} \varphi(\boldsymbol{x}_i) = 0. \tag{3.56}$$

A procedure to select $\boldsymbol{\phi}$ satisfying (3.56) is given in [41, 42]. The PCs can then be computed by solving the eigenvalue problem [41, 42]

$$\lambda \boldsymbol{v} = \boldsymbol{C}_1 \boldsymbol{v} = \frac{1}{N} \sum_{j=1}^{N} \left(\varphi(\boldsymbol{x}_i)^{\mathrm{T}} v \right) \varphi(\boldsymbol{x}_i). \tag{3.57}$$

Thus, $\boldsymbol{v}$ must be in the span of the mapped data

$$\boldsymbol{v} = \sum_{i=1}^{N} \alpha_i \varphi(\boldsymbol{x}_i). \tag{3.58}$$

After premultiplying both sides of (3.58) by $\boldsymbol{\phi}(\mathbf{x}_j)$ and performing mathematical manipulations, the kernel PCA problem reduces to

$$\boldsymbol{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}, \tag{3.59}$$

where λ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)^{\mathrm{T}}$ are, respectively, the eigenvalues and the associated eigenvectors of $\boldsymbol{K}$, and $\boldsymbol{K}$ is an $N \times N$ kernel matrix with

$$\boldsymbol{K}_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi^T(\boldsymbol{x}_i)\varphi(\boldsymbol{x}_j), \tag{3.60}$$

where $\kappa(\cdot)$ is a kernel function.

Popular kernel functions used in the kernel method are the polynomial, Gaussian kernel, and sigmoidal kernels, which are, respectively, given by

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j + \theta\right)^{a0}, \tag{3.61}$$

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathrm{e}^{-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}}, \tag{3.62}$$

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh\left(c_0\left(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j\right) + \theta\right), \tag{3.63}$$

where a_0 is a positive integer, $\sigma > 0$, and c_0, $\theta \in R$. Even if the exact form of $\phi(\cdot)$ does not exist, any symmetric function $\kappa\,(\boldsymbol{x}_i, \boldsymbol{x}_j)$ satisfying Mercer's theorem can be used as a kernel function.

Arrange the eigenvalues in the descending order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{J2} > 0$ and denote their associated eigenvectors as $\boldsymbol{\alpha}_1, \ldots \boldsymbol{\alpha}_{J2}$. The eigenvectors are further normalized as $\boldsymbol{\alpha}_k^T\boldsymbol{\alpha}_k = 1/\lambda_k$.

The nonlinear PCs of $\boldsymbol{x}$ can be extracted by projecting the mapped pattern $\boldsymbol{\phi}(\boldsymbol{x})$ onto $\boldsymbol{v}_k$

$$\boldsymbol{v}_k^{\mathrm{T}}\varphi(\boldsymbol{x}) = \sum_{j=1}^{N} \alpha_{k,j}\kappa(\boldsymbol{x}_j, \boldsymbol{x}), \tag{3.64}$$

for $k = 1, 2, \ldots, J_2$, where $\alpha_{k,\,j}$ is the jth element of $\boldsymbol{\alpha}_k$.

The kernel PCA algorithm is much more complicated and may sometimes be trapped more easily into local minima. The PCA needs to deal with an eigenvalue problem of a $J_1 \times J_1$ matrix, while the kernel PCA needs to solve an eigenvalue problem of an $N \times N$ matrix. Sparse approximation methods can be applied to reduce the computational cost.

3.7.2 Robust/Nonlinear Principal Component Analysis

In order to increase the robustness of the PCA against outliers, a simple way is to eliminate the outliers or replace them by more appropriate values. A better alternative is to use a robust version of the covariance matrix based on the *M*-estimator. The data from which the covariance matrix is constructed may be weighted such that the samples far from the mean have less importance.

Several popular PCA algorithms have been generalized into robust versions by applying a statistical physics approach [43], where the defined objective function can be regarded as a soft generalization of the *M*-estimator. In this subsection, robust PCA algorithms are defined so that the optimization criterion grows less than quadratically and the constraint conditions are the same as for the PCA algorithms [44], which are based on a quadratic criterion. The robust PCA problem usually leads to mildly nonlinear algorithms, in which the nonlinearities appear at selected places only and at least one neuron produces the linear response $y_i = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}_i$. When all neurons generate nonlinear responses $y_i = \varphi(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}_i)$, the algorithm is referred to as the nonlinear PCA.

Variance Maximization-based Robust Principal Component Analysis:

The PCA is to maximize the output variances $E\left[y_i^2\right] = E\left[\left(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}\right)^2\right] = \boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{C}\boldsymbol{w}_i$ of the linear network under orthonormality constraints. In the hierarchical case, the constraints take the form $\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{w}_j = \delta_{ij}, j \le i, \delta_{ij}$ being the Kronecker delta. In the symmetric case, symmetric orthonormality constraints $\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{w}_j = \delta_{ij}$ are applied. The SLA and GHA algorithms correspond to the symmetric and hierarchical network structures, respectively.

To derive robust PCA algorithms, the variance maximization criterion is generalized as $E\left[\sigma\left(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}\right)\right]$ for the *i*th neuron, subject to hierarchical or symmetric orthonormality constraints, where $\sigma(x)$ is the *M*-estimator assumed to be a valid differentiable cost function that grows less than quadratically, at least for large *x*. Examples of such functions are $\sigma(x) = \ln\cosh(x)$ and $\sigma(x) = |x|$. The robust PCA in general does not coincide with the corresponding PCA solution, although it can be close to it. The robust PCA is derived by applying the gradient descent method [21, 44]

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)\varphi(y_i(t))\boldsymbol{e}_i(t), \tag{3.65}$$

$$\boldsymbol{e}_i(t) = \boldsymbol{x}(t) - \hat{\boldsymbol{x}}_i(t), \tag{3.66}$$

$$\hat{\boldsymbol{x}}_i(t) = \sum_{j=1}^{I(i)} y_j(t)\boldsymbol{w}_j(t), \tag{3.67}$$

where $\boldsymbol{e}_i(t)$ is the instantaneous representation error vector, and the influence function $\varphi(\boldsymbol{x}) = \mathrm{d}\sigma(\boldsymbol{x})/\mathrm{d}\boldsymbol{x}$.

In the symmetric case, $I(i) = J_2$ and the errors $\boldsymbol{e}_i(t) = \boldsymbol{e}(t)$, $i = 1, \ldots, J_2$. When $\phi(x) = x$, the algorithm is simplified to the SLA. Otherwise, it defines a robust generalization of Oja's rule, first proposed quite heuristically. In the hierarchical case, $I(i) = i$, $i = 1, \ldots, J_2$. If $\phi(x) = x$, the algorithm coincides exactly with the GHA; Otherwise, it defines a robust generalization of the GHA. In the hierarchical case, $\mathbf{e}_i(t)$ can be calculated in a recursive form $\boldsymbol{e}_i(t) = \boldsymbol{e}_{i-1}(t) - y_i(t)\boldsymbol{w}_i(t)$, with $\boldsymbol{e}_0(t) = \boldsymbol{x}(t)$.

Mean Squared Error Minimization-based Robust Principal Component Analysis:

PCA algorithms can also be derived by minimizing the MSE $E\left[\|\boldsymbol{e}_i\|^2\right]$, where $\boldsymbol{e}_i(t) = \boldsymbol{x}(t) - \hat{\boldsymbol{x}}_i(t)$. Accordingly, robust PCA can be obtained by minimizing $\boldsymbol{1}^{\mathrm{T}} E[\sigma(\boldsymbol{e}_i)] = E\left[\|h(\boldsymbol{e}_i)\|^2\right]$, where $\boldsymbol{1}$ is a J_2-dimensional vector, all of whose entries are unity, and $\sigma(\cdot)$ and $h(\cdot)$ are applied componentwise on the input vector. Here, $h(x) = \sqrt{\sigma(x)}$. When $\sigma(x) = x^2$, it corresponds to the MSE. A robust PCA is defined if $\sigma(x)$ grows less than quadratically. Using the gradient descent method leads to

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)\left[\boldsymbol{w}_i(t)^{\mathrm{T}}\varphi(\boldsymbol{e}_i(t))\boldsymbol{x}(t) + \boldsymbol{x}^{\mathrm{T}}(t)\boldsymbol{w}_i(t)\varphi(\boldsymbol{e}_i(t))\right], \tag{3.68}$$

where $\boldsymbol{w}_i$ estimates the robust counterparts of the principal eigenvectors $\boldsymbol{c}_i$. The first term in the bracket is very small and can be neglected, and thus we can get a simplified algorithm

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)\boldsymbol{x}^{\mathrm{T}}(t)\boldsymbol{w}_i(t)\varphi(\boldsymbol{e}_i(t)) = \boldsymbol{w}_i(t) + \eta(t)y_i(t)\varphi(\boldsymbol{e}_i(t)). \tag{3.69}$$

Algorithms (3.69) and (3.65) resemble each other. However, Algorithm (3.69) generates a linear final input–output mapping, while in Algorithm (3.65) the input–output mapping is nonlinear. When $\phi(x) = x$, algorithms (3.69) and (3.65) are the same as the SLA in the symmetric case, and the same as the GHA in the hierarchical case.

Another Nonlinear Extension to Principal Component Analysis:

A nonlinear PCA algorithm may be derived by the gradient descent method for minimizing the MSE $E\left[\|\boldsymbol{\varepsilon}_i\|^2\right]$, where the error vector $\boldsymbol{\varepsilon}_i$ is a nonlinear extension to $\boldsymbol{e}_i(t) = \boldsymbol{x}(t) - \hat{\boldsymbol{x}}_i(t)$. The nonlinear PCA so obtained has a form similar to the robust PCA given by (3.65) through (3.67)

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \eta(t)\varphi(y_i(t))\boldsymbol{\varepsilon}_i(t), \tag{3.70}$$

$$\boldsymbol{\varepsilon}_i(t) = \boldsymbol{x}(t) - \sum_{j=1}^{I(i)} \varphi(y_j(t))\boldsymbol{w}_j(t), \tag{3.71}$$

for $i = 1, \ldots, J_2$.

In this case, $I(i) = J_2$ and all $\boldsymbol{\varepsilon}_i(t)$ are the same. The nonlinear PCA in the hierarchical case is a direct nonlinear generalization of the GHA. In the hierarchical case, $I(i) = i$ and (3.71) can be computed recursively

$$\boldsymbol{\varepsilon}_i(t) = \boldsymbol{\varepsilon}_{i-1}(t) - \varphi(y_i(t))\boldsymbol{w}_i(t), \tag{3.72}$$

with $\boldsymbol{\varepsilon}_0(t) = \boldsymbol{x}(t)$.

It has been pointed out in [44] that robust and nonlinear PCA algorithms have better stability than the corresponding PCA algorithms if the (odd) nonlinearity $\phi(x)$ grows less than linearly, namely $|\phi(x)| < |x|$. On the contrary, nonlinearities growing faster than linearly cause stability problems easily and therefore are not recommended.

3.7.3 *Autoassociative Network-Based Nonlinear PCA*

The MLP can be used to perform nonlinear dimension reduction and hence nonlinear PCA. Both the input and output layers of the MLP have J_1 units, and one of its hidden layers, known as the bottleneck or representation layer, has J_2 units, $J_2 < J_1$. The network is trained to reproduce its input vectors. This kind of network is called the autoassociative MLP. After the network is trained, it performs a projection onto the J_2-dimensional subspace spanned by the first J_2 principal components of the data. The vectors of weights leading to the hidden units form a basis set that spans the principal subspace, and data compression therefore occurs in the bottleneck layer. Many applications of the MLP in autoassociative mode for PCA are available in the literature [45, 46].

The three-layer autoassociative J_1-J_2-J_1 feedforward network or MLP network can also be used to extract the first J_2 principal components of J_1-dimensional data. If nonlinear activation functions are applied in the hidden layer, the network performs as a nonlinear PCA network. In the case of nonlinear units, local minima certainly appear. However, if linear units are used in the output layer, nonlinearity in the hidden layer is theoretically meaningless [45]. This is due to the fact that the network tries to approximate a linear mapping.

3.8 Other PCA or Extensions of PCA

Besides the algorithms reviewed in the preceding parts, there exist lots of other PCAs or their extensions. For example, there are minor component analysis, constrained PCA, localized PCA, incremental PCA, supervised PCA, complex-valued PCA, two-dimensional PCA, generalized eigenvalue decomposition, singular value decomposition, canonical correlation analysis, etc. Among these algorithms, minor component analysis (MCA), generalized eigenvalue decomposition, and singular

value decomposition are important PCA algorithms or extensions. So we will have separate chapters to study them, respectively. See Chaps. 4, 8 and 9 for more details. Here, we only discuss the remaining algorithms.

Constrained PCA: When certain subspaces are less preferred than others, this yields the constrained PCA [47]. The optimality criterion for constrained PCA is variance maximization, as in PCA, but with an external subspace orthogonality constraint that extracts principal components orthogonal to some undesired subspace [3]. Constrained PCA first decomposes the data matrix by projecting the data matrix onto the spaces spanned by matrices of external information and then applies PCA to the decomposed matrices, which involves generalized SVD. APEX can be applied to recursively solve the constrained PCA problem [26].

Localized PCA: The nonlinear PCA problem can be overcome using localized PCA [3]. First, the data space is partitioned into a number of disjunctive regions, followed by the estimation of the principal subspace within each partition by linear PCA. The distribution is then collectively modeled by a collection of linear PCA models, each characterizing a partition. It should be noted that the localized PCA is different from local PCA. In the latter, the update at each node makes use of only local information. VQ-PCA [48] is a locally linear model that uses vector quantization to define the Voronoi regions for localized PCA. An online localized PCA algorithm [49] was developed by extending the neural gas method. ASSOM is another localized PCA for unsupervised extraction of invariant local features from the input data. Localized PCA provides an efficient means to decompose high-dimensional data compression problems into low-dimensional ones [3].

Incremental PCA: Incremental PCA algorithm can update eigenvectors and eigenvalues incrementally. It is applied to a single training sample at a time, and the intermediate eigen problem must be solved repeatedly for every training sample [50]. Chunk incremental PCA [51] processes a chunk of training samples at a time. It can reduce the training time effectively and obtain major eigenvectors with fairly good approximation. In Chunk incremental PCA, the update of an eigen space is completed by performing single eigenvalue decomposition. The SVD updating-based incremental PCA algorithm [52] gives a close approximation to the batch-mode PCA method, and the approximation error is proved to be bounded. Candid covariance-free IPCA [53] is a fast incremental PCA algorithm, which is used to compute the principal components of a sequence of samples incrementally without estimating the covariance matrix.

Supervised PCA: Like supervised clustering, supervised PCA [54] is achieved by augmenting the input of PCA with the class label of the data set. Class-augmented PCA [55] is a supervised feature extraction method, which is composed of processes for encoding the class information, augmenting the encoded information to data, and extracting features from class-augmented data by applying PCA.

Complex-valued PCA: Complex PCA is a generalization of PCA in complex-valued data sets [56], and it employs the same neural network architecture as for PCA, but with complex weights. Complex-domain GHA [57] extends GHA for complex principal component extraction, and it is very similar to GHA except

that complex notations are introduced. In [58], a complex-valued neural network, model is developed for nonlinear complex PCA, and it uses the architecture of Kramer's nonlinear PCA network, but with complex weights and biases. The algorithm can extract nonlinear features missed by PCA. Both PAST and PASTd are, respectively, the PSA and PCA algorithms derived for complex-valued signals [28]. Complex-valued APEX [59] actually allows for extracting a number of principal components from a complex-valued signal. The robust complex PCA algorithms have also been derived in [60] for hierarchically extracting principal components of complex-valued signals using a robust statistics-based loss function.

Two-dimensional PCA: Because of the small-sample-size problem for image representation, PCA is prone to be overfitted to the training set. Two-dimensional PCA can address these problems. In two-dimensional PCA, an image covariance matrix is constructed directly using the original image matrices instead of the transformed vectors, and its eigenvectors are derived for image feature extraction.

2DPCA [61] evaluates the covariance (scatter) matrix more accurately than PCA does, since it only reflects the information between rows and is a row-based PCA. Diagonal PCA [62] improves 2DPCA by defining the image scatter matrix as the covariances between the variations of the rows and those of the columns of the images and is more accurate than PCA and 2DPCA. In modular PCA [63], an image is divided into n_1 subimages and PCA is performed on all these subimages. 2DPCA and modular PCA both solve the overfitting problems by reducing the dimension and by increasing the training vectors yet introduce the high feature dimension problem.

Bidirectional PCA [64] reduces the dimension in both column and row directions for image feature extraction, whose feature dimension is much less than that of 2DPCA. It has to be performed in batch mode. PCA-L_1 [65] is a fast and robust L_1-norm-based PCA. L_1-norm-based two-dimensional PCA (2DPCA-L_1) [66] is a two-dimensional generalization of PCA-L_1 [65]. It avoids the eigen decomposition process, and its iteration step is easy to perform. The uncorrelated multilinear PCA algorithm [67] is used for unsupervised subspace learning of tensorial data. It not only obtains features that maximize the variance captured, but also enforces a zero-correlation constraint, thus extracting uncorrelated features.

3.9 Summary

An overview of a variety of neural network-based principal component analysis algorithms has been presented in this chapter. Many new adaptive PCA algorithms are being added to this field, indicating a consistent interest in this direction. Nevertheless, neural network-based PCA algorithms have been considered a matured subject. Many problems and current research interest lie in performance analysis of PCA algorithms, minor component analysis, generalization or extensions of PCA algorithms, etc., which will be discussed in the next chapters.

References

1. Diamantaras, K. I., & Kung, S. Y. (1996). *Principal component neural networks: Theory and applications*. New York: Wiley.
2. Jolliffe, I. T. (2002). *Principal component analysis* (2nd ed.). Berlin: Springer.
3. Du, K. L., & Swamy, M. N. S. (2013). *Neural networks and statistical learning*. Berlin: Springer.
4. Liu J. L, Wang H., Lu J. B., Zhang B. B., & Du K. L. (2012). Neural network implementations for PCA and its extensions. *Artificial Intelligence*. doi:10.5402/2012/847305
5. Hebb, D. O. (1949). *The organization of behavior*. New York: Wiley.
6. Zafeiriou, S., & Petrou, M. (2010). Nonlinear non-negative component analysis algorithms. *IEEE Transactions on Image Processing, 19*(4), 1050–1066.
7. Rubner, J., & Tavan, P. (1989). A self-organizing network for principal-component analysis. *Europhysics Letters, 10*(7), 693–698.
8. Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology, 15*(3), 267–273.
9. Yuille, A. L., Kammen, D. M., & Cohen, D. S. (1989). Quadrature and development of orientation selective cortical cells by Hebb rules. *Biological Cybernetics, 61*(3), 183–194.
10. Linsker, R. (1986). From basic network principles to neural architecture. *Proceedings of the National Academy of Sciences of the USA* (Vol. 83, pp. 7508–7512), 8390-8394, 9779-8783.
11. Linsker, R. (1988). Self-organization in a perceptual network. *IEEE Computer, 21*(3), 105–117.
12. Hassoun, M. H. (1995). *Fundamentals of artificial neural networks*. Cambridge, MA: MIT Press.
13. Ljung, L. (1977). Analysis of recursive stochastic algorithm. *IEEE Transactions on Automatic Control, 22*(4), 551–575.
14. Zufiria, P. J. (2002). On the discrete-time dynamics of the basic Hebbian neural-network node. *IEEE Transactions on Neural Networks, 13*(6), 1342–1352.
15. Zhang, Y., Ye, M., Lv, J. C., & Tan, K. K. (2005). Convergence analysis of a deterministic discrete system of Oja's PCA learning algorithm. *IEEE Transactions on Neural Networks, 16*(6), 1318–1328.
16. Oja, E., & Karhunen, J. (1985). On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications, 106*(1), 69–84.
17. Chen, L. H., & Chang, S. (1995). An adaptive learning algorithm for principal component analysis. *IEEE Transactions on Neural Networks, 6*(5), 1255–1263.
18. Oja, E. (1992). Principal components, minor components, and linear neural networks. *Neural Networks, 5*(6), 929–935.
19. Oja, E., Ogawa, H., & Wangviwattana, J. (1992). Principal component analysis by homogeneous neural networks. *IEICE Transactions on Information and Systems, 75*(3), 366–382.
20. Sanger, T. D. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks, 2*(6), 459–473.
21. Xu, L. (1993). Least mean square error reconstruction principle for self-organizing neural-nets. *Neural Networks, 6*(5), 627–648.
22. Rubner, J., & Schulten, K. (1990). Development of feature detectors by self-organization. *Biological Cybernetics, 62*(62), 193–199.
23. Rubner, J., & Tavan, P. (1989). A self-organizing network for principal-component analysis. *Europhysics Letters, 10*(7), 693–698.
24. Kung, S. Y., & Diamantaras, K. I. (1990). A neural network learning algorithm for adaptive principal components extraction (APEX). *Proceedings of IEEE ICCASSP* (pp. 861–864). Albuquerque, NM.

25. Foldiak, P. (1989). Adaptive network for optimal linear feature extraction. *Proceedings of International Joint Conference Neural Networks (IJCNN)* (Vol. 1, pp. 401–405). Washington, DC.
26. Kung, S. Y., Diamantaras, K. I., & Taur, J. S. (1994). Adaptive principal components extraction (APEX) and applications. *IEEE Transactions on Signal Processing, 42*(5), 1202–1217.
27. Fiori, S., & Piazza, F. (1998). A general class of ψ-APEX PCA neural algorithms. *IEEE Transactions on Circuits and Systems I, 47*(9), 1394–1397.
28. Yang, B. (1995). Projection approximation subspace tracking. *IEEE Transactions on Signal Processing, 43*(1), 95–107.
29. Bannour, S., & Azimi-Sadjadi, M. R. (1995). Principal component extraction using recursive least squares learning. *IEEE Transactions on Neural Networks, 6*(2), 457–469.
30. Miao, Y., & Hua, Y. (1998). Fast subspace tracking and neural network learning by a novel information criterion. *IEEE Transactions on Signal Processing, 46*(7), 1967–1979.
31. Ouyang, S., Bao, Z., & Liao, G. (2000). Robust recursive least squares learning algorithm for principal component analysis. *IEEE Transactions on Neural Networks, 11*(1), 215–221.
32. Ouyang, S., & Bao, Z. (2002). Fast principal component extraction by a weighted information criterion. *IEEE Transactions on Signal Processing, 50*(8), 1994–2002.
33. Chatterjee, C., Roychowdhury, V. P., & Chong, E. K. P. (1998). On relative convergence properties of principal component analysis algorithms. *IEEE Transactions on Neural Networks, 9*(2), 319–329.
34. Yang, B. (1995). An extension of the PASTd algorithm to both rank and subspace tracking. *IEEE Signal Processing Letters, 2*(9), 179–182.
35. Chauvin, Y. (1989). Principal component analysis by gradient descent on a constrained linear Hebbian cell. *Proceedings of the International Joint Conference on Neural Networks* (pp. 373–380). Washington, DC.
36. Fu, Z., & Dowling, E. M. (1995). Conjugate gradient eigenstructure tracking for adaptive spectral estimation. *IEEE Transactions on Signal Processing, 43*(5), 1151–1160.
37. Kang, Z., Chatterjee, C., & Roychowdhury, V. P. (2000). An adaptive quasi-Newton algorithm for eigensubspace estimation. *IEEE Transactions on Signal Processing, 48*(12), 3328–3333.
38. Ouyang, S., Ching, P. C., & Lee, T. (2003). Robust adaptive quasi-Newton algorithms for eigensubspace estimation. *IEEE Proceedings—Vision, Image and Signal Processing, 150*(5), 321–330.
39. Moller, R., & Konies, A. (2004). Coupled principal component analysis. *IEEE Transactions on Neural Network, 15*(1), 214–222.
40. Moller, R. (2006). First-order approximation of Gram-Schmidt orthonormalization beats deflation in coupled PCA learning rules. *Neurocomputing, 69*(13–15), 1582–1590.
41. Schölkopf, B., Smola, A., & Müller, K. R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computations, 10*(5), 1299–1319.
42. Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K. R., Rätsch, G., et al. (1970). Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks, 10*(5), 1000–1017.
43. Xu, L., & Yuille, A. L. (1995). Robust principal component analysis by self-organizing rules based on statistical physics approach. *IEEE Transactions on Neural Networks, 6*(1), 131–143.
44. Karhunen, J., & Joutsensalo, J. (1995). Generalizations of principal component analysis, optimization problems, and neural networks. *Neural Networks, 8*(4), 549–562.
45. Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics, 59*(4–5), 291–294.
46. Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal, 37*(2), 233–243.
47. Kung, S. Y. (1990). Constrained principal component analysis via an orthogonal learning network. *Proceedings of the IEEE International Symposium on Circuits and Systems* (Vol. 1, pp. 719–722). New Orleans, LA.

48. Kambhatla, N., & Leen, T. K. (1993). Fast non-linear dimension reduction. *Proceedings of IEEE International Conference on Neural Networks* (Vol. 3, pp. 1213–1218). San Francisco, CA.
49. Moller, R., & Hoffmann, H. (2004). An extension of neural gas to local PCA. *Neurocomputing, 62*(1), 305–326.
50. Hall, P., & Martin, R. (1998). Incremental eigenanalysis for classification. *Proceedings of British Machine Vision Conference* (Vol. 1, pp. 286–295).
51. Ozawa, S., Pang, S., & Kasabov, N. (2008). Incremental learning of chunk data for online pattern classification systems. *IEEE Transactions on Neural Networks, 19*(6), 1061–1074.
52. Zhao, H., Yuen, P. C., & Kwok, J. T. (2006). A novel incremental principal component analysis and its application for face recognition. *IEEE Transactions on Systems, Man, and Cybernetics, 36*(4), 873–886.
53. Weng, J., Zhang, Y., & Hwang, W. S. (2003). Candid covariance-free incremental principal component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 25*(8), 1034–1040.
54. Chen, S., & Sun, T. (2005). Class-information-incorporated principal component analysis. *Neurocomputing, 69*(1–3), 216–223.
55. Park, M. S., & Choi, J. Y. (2009). Theoretical analysis on feature extraction capability of class-augmented PCA. *Pattern Recognition, 42*(11), 2353–2362.
56. Horel, J. D. (1984). Complex principal component analysis: Theory and examples. *Journal of Climate and Applied Meteorology, 23*(12), 1660–1673.
57. Zhang, Y., & Ma, Y. (1997). CGHAfor principal component extraction in the complex domain. *IEEE Transactions on Neural Networks, 8*(5), 1031–1036.
58. Rattan, S. S. P., & Hsieh, W. W. (2005). Complex-valued neural networks for nonlinear complex principal component analysis. *Neural Networks, 18*(1), 61–69.
59. Chen, Y., & Hou, C. (1992). High resolution adaptive bearing estimation using a complex-weighted neural network. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (Vol. 2, pp. 317–320). San Francisco, CA.
60. Cichocki, A., Swiniarski, R. W., & Bogner, R. E. (2010). Hierarchical neural network forrobust PCA computation of complex valued signals. In *World Congress on Neural Networks* (pp. 818–821).
61. Yang, J., Zhang, D., Frangi, A. F., & Yang, J. Y. (2004). Two-dimensional PCA: A new approach to appearance-based face representation and recognition. *IEEE Transactions on Pattern Analysis Machine Intelligence, 26*(1), 131–137.
62. Zhang, D., Zhou, Z. H., & Chen, S. (2006). Diagonal principal component analysis for face recognition. *Pattern Recognition, 39*(1), 140–142.
63. Gottumukkal, R., & Asari, V. K. (2004). An improved face recognition technique based on modular PCA approach. *Pattern Recognition Letters, 25*(4), 429–436.
64. Zuo, W., Zhang, D., & Wang, K. (2006). Bidirectional PCA with assembled matrix distance metric for image recognition. *IEEE Transactions on Systems, Man, and Cybernetics B, 36*(4), 863–872.
65. Kwak, N. (2008). Principal component analysis based on L1-norm maximization. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 30*(9), 1672–1680.
66. Li, X., Pang, Y., & Yuan, Y. (2010). L1-norm-based 2DPCA. *IEEE Transactions on Systems, Man, and Cybernetics B, 40*(4), 1170–1175.
67. Lu, H., Plataniotis, K. N. K., & Venetsanopoulos, A. N. (2009). Uncorrelated multilinear principal component analysis for unsupervised multilinear subspace learning. *IEEE Transactions on Neural Networks, 20*(11), 1820–1836.