Neural Comput & Applic (2016) 27:1245–1261
DOI 10.1007/s00521-015-1922-z

ORIGINAL ARTICLE

An adaptive neural networks formulation for the two-dimensional principal component analysis

Xianye Ben[1,2] · Weixiao Meng[3] · Kejun Wang[4] · Rui Yan[5]

Received: 28 January 2015 / Accepted: 30 April 2015 / Published online: 12 May 2015

Abstract This study, for the first time, developed an adaptive neural networks (NNs) formulation for the two-dimensional principal component analysis (2DPCA), whose space complexity is far lower than that of its statistical version. Unlike the NNs formulation of principal component analysis (PCA, i.e., 1DPCA), the solution with lower iteration in nature aims to directly deal with original image matrices. We also put forward the consistence in the conceptions of 'eigenfaces' or 'eigengaits' in both 1DPCA and 2DPCA neural networks. To evaluate the performance of the proposed NN, the experiments were carried out on AR face database and on 64 × 64 pixels gait energy images on CASIA(B) gait database. The less reconstruction error was exploited using the proposed NN in the condition of a large sample set compared to adaptive estimation of learning algorithms for NNs of PCA. On the contrary, if the sample set was small, the proposed NN could achieve a higher residue error than PCA NNs. The amount of calculation for the proposed NN here could be smaller than that for the PCA NNs on the feature extraction of the same image matrix, which represented an efficient solution to the problem of training images directly. On face and gait recognition tasks, a simple nearest neighbor classifier test indicated a particular benefit of the neural network developed here which serves as an efficient alternative to conventional PCA NNs.

Keywords Two-dimensional principal component analysis (2DPCA) · Neural network (NN) · Neural networks formulation · Eigenface · Eigengait

✉ Xianye Ben
benxianye@gmail.com

1 School of Information Science and Engineering, Shandong University, Jinan 250100, China

2 Key Laboratory of Intelligent Perception and Systems for High-Dimensional Information, Ministry of Education, Nanjing University of Science and Technology, Nanjing 210094, China

3 School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin 150080, China

4 College of Automation, Harbin Engineering University, Harbin 150001, China

5 Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

1 Introduction

Two-dimensional principal component analysis (2DPCA) [1] is a state-of-the-art statistical technique developed for image representation. As opposed to principal component analysis (PCA, i.e., 1DPCA), 2DPCA is based on 2D matrices rather than 1D vectors, making it unnecessary to transform the image matrix into a vector for feature extraction. Overall, the idea of 2-D method here originates preliminarily from the direct construction of image scatter matrices by using the original image matrices. Besides, the image covariance matrix and image scatter matrices of 2DPCA can have a much smaller size in comparison with its counterpart PCA method. Therefore, 2DPCA significantly reduces the computational cost and avoids the singularity problem [2]. For example, if the image size is 64 × 64 pixels, the image covariance matrix of 2DPCA is still 64 × 64 pixels, regardless of the size of the training sample. As a result, 2DPCA has a remarkable computational advantage over PCA. Its first principal component is a 1D linear subspace where the variance of the data is maximal, and the second principal component is the

direction of maximal variance in the space orthogonal to the first principal component. 1D principal components are computed by PCA; likewise, 2D principal components are computed by 2DPCA.

Research interest in 2DPCA has increased recently [3–5]. 2DPCA is essentially working in the row direction of images. Zhang and Zhou [6] proposed an alternative 2DPCA which worked in the column direction of images, and developed the two-directional 2DPCA considering the row and column directions simultaneously. Ye [7] proposed another version of two-sided linear transformation called generalized low-rank approximations of matrices (GLRAM) as an extension to 2DPCA, but it is an iterative approach. Liu and Chen [8] proposed a non-iterative GLRAM (NIGLRAM) to overcome GLRAM's shortcomings such as lacking a criterion to automatically determine the dimensionalities of the projected matrices. Lu et al. [9] proposed a new simplified version of GLRAM with the purpose of deriving the projection matrices for GLRAM. Kim et al. [10] proposed a face recognition approach using a fusion method based on bidirectional 2DPCA. Yang and Liu [11] presented a bidirectional 2DPCA-based discriminant analysis (HVDA) method for face verification. Although 2D image matrices are used to directly construct the image covariance matrix, these algorithms often run up against computational limits due to the high space complexity for dealing with large image matrices, especially for images and videos. Taking 2DPCA for example, the space complexity for computing the eigendecomposition of an image scatter matrix with the size of $n \times n$ using Jacobi method is $O(n^3)$. As the dimensionality n increases, the fact cannot be ignored that it may outstrip the processing capability of single-chip microcomputer or embedded system. Consequently, the algorithmic solution of 2DPCA based on statistics cannot be used effectively in performing data processing for large-scale images, and other implementations are needed which are able to reduce the space complexity.

During the last decade, a number of researchers have proposed various neural networks (NNs) methods for statistical analysis and machine learning. Details about PCA algorithms can be found in [12, 13]. All the presented neural network approaches for PCA can be systematically derived from the original formulation by Oja [14] of a single neuron with the Hebbian learning principal component analyzer. The single-neuron case then was extended to estimation of several principal components. The single-layer neural network architecture for multiple principal eigenvector extraction was proposed by Oja and Karhunen [15]. These PCA NNs can be described by stochastic discrete-time (SDT) algorithms, and some invariant sets are warranted to be non-divergence of these NNs by choosing proper learning parameters [16]. The generalized Hebbian algorithm (GHA) [17] and the stochastic gradient ascent (SGA) algorithm [18] can be directly derived from a symmetric subspace learning rule. An adaptive principal component extraction algorithm was presented by Kung et al. [19]. These five neural network approaches for PCA can be classified into two categories: reestimation algorithms and decorrelating algorithms [20]. Due to PCA's locality, it has been argued that these algorithms are 'biologically plausible.' Andreas and Kurt [21] proposed the local PCA algorithms and fully described their equivalence, where all lateral connections are set to zero along with their local stability. Kong et al. [22] used a deterministic discrete-time (DDT) system to analyze the convergence of a unified PCA and minor component analysis (MDA) algorithm. Karhunen et al. [23] introduced learning algorithms for each of the three layers of the proposed independent component analysis (ICA) network to be used for blind source separation. Gou and Jiao presented a method for texture image recognition using a synergetic neural network (SNN) combined with immune clonal strategy (ICS) and fuzzy clustering to train the prototype vectors; this method was used to classify object images into groups [24]. Training a radial basis function (RBF) network consisting of three layers, input, hidden and output layers, to be a classifier with computing efficiency means optimizing the parameters of centers, widths and weights in the network. For off-line training, the K-means, the P-nearest neighbors and the batch least squares (BLS) algorithms are used. When the classifier is used online, the centers remain fixed, as they have been chosen to be distributed in the whole operating space, while the widths and weights are adapted to minimize the classification error caused by any time-varying dynamics and model uncertainty. The widths are adapted using a gradient descent algorithm, and the weights are adapted using the recursive least squares (RLS) algorithm [25]. Tomenko [26] proposed an online nonlinear dimensionality reduction using competitive learning and RBF. In order to achieve scalability, he used modified topology to represent networks and geodesic distance and estimated sampled or streaming data with a finite set of reference patterns. Kohonen networks are well known for unsupervised learning cluster analysis. A fuzzy Kohonen clustering network was proposed, which integrated the fuzzy c-means (FCM) model into the learning rate and updating strategies of the Kohonen network. This yielded an optimization problem related to FCM [27]. Ceylan et al. [28] presented a comparative study of four different structures: FCM NN, PCA NN, FCM–PCA NN and WT NN (wavelet transform). Huang et al. [29] designed the hybrid RBF NNs realized with FCM and polynomial NN. FCM was employed to

defeat a possible curse of dimensionality, and polynomial NN was used to build local models. Alexandridis and Zapranis [30] proposed a complete statistical model identification framework to apply wavelet NNs. Those issues were soundly studied: their structures, training methods, initialization algorithms, variable significance, variable selection algorithms, model selection methods and constructing confidence and prediction intervals methods. Zhang et al. [31] applied a symmetric NN to learn the features of a data by minimizing the reconstruction error between the encoder layer's input data and the decoding layer's reconstruction data. Carvajal and Figueroa [32] presented analog adaptive linear combiners and on-chip learning for least mean square and generalized Hebbian algorithm. Recent years have brought significant improvements in statistical analysis in real-world settings [33–35]. The advantage of these aforementioned neural networks (NNs) methods for statistical analysis and machine learning is online learning, which is necessary if not all training patterns are available all the time. Besides, time-varying delays systems often exist in the system output of NN [36–38].

Motivated by the aforementioned neural network implementation of statistical algorithms, especially Hebbian learning and adaptive principal component extraction [14, 17, 19], we will investigate a more challenging problem in this paper, namely an adaptive neural networks formulation for the 2DPCA. It is also an online learning implementation. This NN is based on Hebbian learning and adaptive principal component extraction; however, it deviates from the previous research. Because the proposed NNs can directly deal with original image matrices, accomplished by several time-varying delay units.

The two major difficulties lie in the fact of how to design the architecture of this NN and estimate several principal components using this network. The main contributions are as follows.

1. A new neuron model is introduced to solve the problem of adaptively estimating the first principal component in order to directly deal with original image matrices. The attributes of its weight vector when the network converges are discussed.
2. A new neuron model for adaptively estimating several principal components is proposed. Learning steps of estimation of several principal components are then presented. Moreover, its space complexity is far lower than that of standard 2DPCA based on statistics.
3. The conceptions of 'eigenfaces' for 2DPCA neural network is put forward for the first time, and 2DPCA 'eigenfaces' have produced results essentially in agreement with PCA 'eigenfaces'.

The remainder of this paper is organized as follows. Section 2 gives adaptive estimation of the first principal component for 2DPCA. Section 3 describes adaptive estimation of several principal components for 2DPCA. Performance analysis and simulation results are given in Sect. 4, and then, conclusions are provided in Sect. 5.

2 Adaptive estimation of the first principal component

2.1 Neuron model

As shown in Fig. 1, the input to the synapses is a matrix signal $\boldsymbol{X} \in R^{m\times n}$, with the individual vector components given as x_{1j}, x_{2j}, …, x_{mj}, for $j = 1, 2, \ldots, n$, and then, the expression of $\boldsymbol{X}$ is:

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$$

Therefore, the number of times for inputting a matrix signal is n. Namely, the first, second, etc., inputs are, respectively, the first, second, etc., column vector of matrix signal $\boldsymbol{X}$. Each component x_{ij}, for $i = 1, 2, \ldots, m$, is multiplied by the weight w_i in the form of a linear activation function. Thus, the output of the network is written as

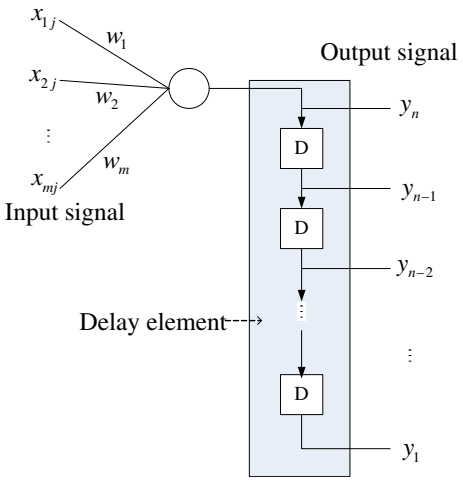

Fig. 1 Neuron model for estimating the first principal component. The delay element consists of n unit-delay sub-operators with the function of the shift storage. It is also called an ordinary tapped delay line memory of order n. The jth ($j = 1, \ldots, n$) column of the image (matrix) is the input, and the output is a scalar y_j. Similarly, the $(j + 1)$th column of the image (matrix) is the input, and y_{j+1} is the corresponding output; at the same time, y_j is shifted down to another storage unit. Therefore, all columns of the image (matrix) are regarded as its input, and the output is a vector $[y_1, y_2, \ldots, y_n]$

$$y_j = \sum_{i=1}^{m} w_i x_{ij} = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_j \tag{1}$$

where $\boldsymbol{x}_j$ is the jth column vector of matrix signal $\boldsymbol{X}$. Here, y_j, for $j = 1, 2, \ldots, n$, is ordered as

$$\boldsymbol{y} = [y_1, y_2, \ldots, y_n] \tag{2}$$

Now, after the network architecture is obtained, the method of adjusting the weight vector $\boldsymbol{w}$ is discussed as follows:

Define an objective function as

$$\begin{aligned} f(\boldsymbol{w}) &= \frac{\mathrm{E}(\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}})}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}} \\ &= \frac{\mathrm{E}[(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X})(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X})^{\mathrm{T}}]}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}} \\ &= \frac{\mathrm{E}[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w}]}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}} \\ &= \frac{\boldsymbol{w}^{\mathrm{T}}\mathrm{E}[\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}]\boldsymbol{w}}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}} \\ &= \frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w}}{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}} \end{aligned} \tag{3}$$

where $\boldsymbol{R} = \mathrm{E}[\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}]$ is termed as the covariance matrix, and E[] denotes the operator expectation which means an average over the 'training set.' It maximizes the decreasing rate of the covariance directly based on matrix processing. Although the object function of 2DPCA is very similar as that of PCA since they are both useful for reducing the dimensionality of data with minimal loss of information, there is a key difference in their covariance matrices, respectively, constructed by matrix and vector data. In addition, the output of PCA is a numerical value, whereas the proposed 2DPCA neural network gives rise to a vector result that is the feature extracted of the input matrix data.

To obtain the weight vector $\boldsymbol{w}$ when $f(\boldsymbol{w})$ is maximized, we can take its partial derivative with respect to $\boldsymbol{w}$, namely

$$\nabla f = \frac{2\boldsymbol{R}\boldsymbol{w}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}) - (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w})2\boldsymbol{w}}{(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w})^2} \tag{4}$$

Noting that the unit length of the vector $\boldsymbol{w}$ can be expressed through the L_2 norm as $\|\boldsymbol{w}\|_2^2 = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} = 1$, we can write

$$\begin{aligned} \nabla f &= 2\boldsymbol{R}\boldsymbol{w} - (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w})2\boldsymbol{w} \\ &= 2\mathrm{E}[\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}]\boldsymbol{w} - 2\mathrm{E}(\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}})\boldsymbol{w} \end{aligned} \tag{5}$$

After replacing $\mathrm{E}[\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}]$ and $\mathrm{E}(\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}})$ with certain samples, we can rewrite Eq. (5) as follows:

$$\begin{aligned} \nabla f &= 2\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - 2\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{w} \\ &= 2\boldsymbol{X}\boldsymbol{y}^{\mathrm{T}} - 2\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{w} \end{aligned} \tag{6}$$

The learning rule of 2DPCA is given as

$$\Delta\boldsymbol{w} = \eta(\boldsymbol{X}\boldsymbol{y}^{\mathrm{T}} - \boldsymbol{y}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{w}) \tag{7}$$

where $\eta > 0$ is the learning rate parameter. Therefore, the rule of adjusting the weight vector $\boldsymbol{w}$ of this network is

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta(\boldsymbol{X}(t)y^T(t) - y(t)y^T(t)\boldsymbol{w}(t)) \tag{8}$$

where t denotes discrete time.

The network generally converges within several times as the weight vector $\boldsymbol{w}$ is adjusted by Eq. (8).

In summary, differences can be found between the proposed 2DPCA NN and the PCA NN and are as follows: ① different network structures; ② different learning rules of weights and ③ different information processing characteristics of neurons.

2.2 Properties of the weight vector

When converged, the 2DPCA network has the following conclusions,

1. $\|\boldsymbol{w}\|^2 = 1$.

The expectation of the adjusting value of the weight vector $\boldsymbol{w}$ is assumed to be equal to zero when the network has converged, that is,

$$\begin{aligned} 0 &= \frac{\mathrm{E}(\Delta\boldsymbol{w})}{\eta} = \mathrm{E}(\boldsymbol{X}y^T - \boldsymbol{y}\boldsymbol{y}^T w) \\ &= \mathrm{E}(\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{w} - \boldsymbol{y}\boldsymbol{y}^T\boldsymbol{w}) = \boldsymbol{R}\boldsymbol{w} - (\boldsymbol{w}^T\boldsymbol{R}\boldsymbol{w})\boldsymbol{w} \end{aligned} \tag{9}$$

Thus,

$$\boldsymbol{R}\boldsymbol{w} = (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w})\boldsymbol{w} \tag{10}$$

where $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w}$ is a numeric value, and it is the coefficient of $\boldsymbol{w}$. Let $\lambda = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w}$, therefore, $\boldsymbol{R}\boldsymbol{w} = \lambda\boldsymbol{w}$, where $\boldsymbol{w}$ denotes the eigenvector of $\boldsymbol{R}$, and λ is the eigenvalue of $\boldsymbol{R}$. Hence,

$$\lambda = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w} = \boldsymbol{w}^{\mathrm{T}}\lambda\boldsymbol{w} = \lambda\|\boldsymbol{w}\|^2 \tag{11}$$

Thus, the weight vector $\boldsymbol{w}$ has a unit length, that is, $\|\boldsymbol{w}\|^2 = 1$.

2. $\boldsymbol{w}$ lies in the direction of the eigenvector corresponding to the largest eigenvalue.

Let $\boldsymbol{\varphi}_1$ denote one normalized eigenvector of the covariance matrix $\boldsymbol{R}$, that is,

$$\boldsymbol{R}\boldsymbol{\varphi}_1 = \lambda_1\boldsymbol{\varphi}_1 \tag{12}$$

where $\|\boldsymbol{\varphi}_1\| = 1$. When the network converges, $\boldsymbol{w}$ approached $\boldsymbol{\varphi}_1$, namely:

$$\boldsymbol{w} = \boldsymbol{\varphi}_1 + \boldsymbol{\delta} \tag{13}$$

where $\boldsymbol{\delta}$ is disturbing item.

Alternatively, (13) can be expressed by

$$\Delta \boldsymbol{w} = \Delta \boldsymbol{\delta} \tag{14}$$

where Δ indicates an increment, or

$$\mathrm{E}(\Delta \boldsymbol{\delta}) = \mathrm{E}(\Delta \Delta \boldsymbol{w}) = \eta \mathrm{E}(\nabla f) \tag{15}$$

Because of $\eta > 0$ which is not affect the direction of eigenvector, η is omitted. Then, $\mathrm{E}(\Delta \boldsymbol{\delta})$ can be evaluated by

$$\begin{aligned} \mathrm{E}(\Delta \boldsymbol{\delta}) &= \boldsymbol{R}\boldsymbol{w} - (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w})\boldsymbol{w} \\ &= \boldsymbol{R}(\boldsymbol{\varphi}_1 + \boldsymbol{\delta}) - (\boldsymbol{\varphi}_1 + \boldsymbol{\delta})^{\mathrm{T}}\boldsymbol{R}(\boldsymbol{\varphi}_1 + \boldsymbol{\delta})(\boldsymbol{\varphi}_1 + \boldsymbol{\delta}) \\ &= \boldsymbol{R}\boldsymbol{\varphi}_1 + \boldsymbol{R}\boldsymbol{\delta} - (\boldsymbol{\varphi}_1^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\varphi}_1 + \boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\varphi}_1 + \boldsymbol{\varphi}_1^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\delta} + \boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\delta})(\boldsymbol{\varphi}_1 + \boldsymbol{\delta}) \end{aligned} \tag{16}$$

The quadratic of $\boldsymbol{\delta}$ is omitted, and then,

$$\begin{aligned} \mathrm{E}(\Delta\delta) &= \boldsymbol{R}\boldsymbol{\varphi}_1 + \boldsymbol{R}\boldsymbol{\delta} - (\boldsymbol{\varphi}_1^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\varphi}_1 + \boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\varphi}_1 + \boldsymbol{\varphi}_1^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\delta})(\boldsymbol{\varphi}_1 + \boldsymbol{\delta}) \\ &= \boldsymbol{R}\boldsymbol{\delta} - \lambda_1\boldsymbol{\delta} - 2\lambda_1[\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\varphi}_1]\boldsymbol{\varphi}_1 \end{aligned} \tag{17}$$

Assume that it exists another normalized eigenvector $\boldsymbol{\varphi}_2$ of $\boldsymbol{R}$, $\boldsymbol{\varphi}_1 \neq \boldsymbol{\varphi}_2$. Our idea is to project $\mathrm{E}(\Delta\delta)$ onto $\boldsymbol{\varphi}_2$ by the following linear transformation

$$\begin{aligned} \boldsymbol{\varphi}_2^{\mathrm{T}}\mathrm{E}(\Delta\delta) &= \boldsymbol{\varphi}_2^{\mathrm{T}}(\boldsymbol{R}\boldsymbol{\delta} - \lambda_1\boldsymbol{\delta} - 2\lambda_1[\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\varphi}_1]\boldsymbol{\varphi}_1) \\ &= \boldsymbol{\varphi}_2^{\mathrm{T}}\boldsymbol{R}\boldsymbol{\delta} - \boldsymbol{\varphi}_2^{\mathrm{T}}\lambda_1\boldsymbol{\delta} - \boldsymbol{\varphi}_2^{\mathrm{T}}2\lambda_1[\boldsymbol{\delta}^{\mathrm{T}}\boldsymbol{\varphi}_1]\boldsymbol{\varphi}_1 \\ &= \lambda_2\boldsymbol{\varphi}_2^{\mathrm{T}}\boldsymbol{\delta} - \lambda_1\boldsymbol{\varphi}_2^{\mathrm{T}}\boldsymbol{\delta} \\ &= (\lambda_2 - \boldsymbol{\lambda_1})\boldsymbol{\varphi}_2^{\mathrm{T}}\boldsymbol{\delta} \end{aligned} \tag{18}$$

Discussion:

The first case $\boldsymbol{\varphi}_2^{\mathrm{T}}\boldsymbol{\delta} > 0$.

Namely, the direction where $\boldsymbol{\delta}$ is projected onto $\boldsymbol{\varphi}_2$ is positive. It can be shown that $\boldsymbol{\varphi}_2^{\mathrm{T}}\mathrm{E}[\Delta\delta] > 0$ if the eigenvalues $\lambda_2 > \lambda_1$..

The second case $\boldsymbol{\varphi}_2^{\mathrm{T}}\boldsymbol{\delta} < 0$.

Namely, the direction where $\boldsymbol{\delta}$ is projected onto $\boldsymbol{\varphi}_2$ is negative. It can be shown that $\boldsymbol{\varphi}_2^{\mathrm{T}}\mathrm{E}[\Delta\delta] < 0$ if the eigenvalues $\lambda_2 > \lambda_1$.

To sum up the two cases above, $\mathrm{E}[\Delta\delta]$ always changes toward the positive direction of $\boldsymbol{\varphi}_2$, which means that w always changes toward the direction of the eigenvector corresponding to the larger eigenvalue. Therefore, $\boldsymbol{w}$ locates the direction of the eigenvector of $\boldsymbol{R}$ corresponding to the largest eigenvalue consequentially after convergence of this network.

3. The weight vector $\boldsymbol{w}$ maximizes the variance of the output $\boldsymbol{y} = [y_1, y_2, \ldots, y_n]$, where $y_j = \sum_{i=1}^{m} w_i x_{ij} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_j$. The covariance can be denoted by

$$\mathrm{E}[\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}}] = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w} \tag{19}$$

The unitary vector $\boldsymbol{w}$ that maximizes $\mathrm{E}[\boldsymbol{y}\boldsymbol{y}^{\mathrm{T}}]$ is called the optimal projection axis. When $\boldsymbol{w}$ lies in the direction of the eigenvector of $\boldsymbol{R}$ corresponding to the largest eigenvalue, the quadratic form $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w}$ will be maximized.

Apparently, we can draw the conclusion that the weight vector $\boldsymbol{w}$ converges to the normalized eigenvector of $\boldsymbol{R}$ corresponding to the largest eigenvalue through iterative learning in Eq. (8). Therefore, an $m \times n$ random matrix (or image) is compressed into a vector with the dimension of $1 \times n$. In addition, it is certain that mean square error of compressed results is minimum.

3 Adaptive estimation of several principal components

3.1 Neuron model

The neural network in Fig. 1 is able to obtain the first principal component. In succession, several principal components with a number of output nodes are taken into account, which is illustrated in Fig. 2a.

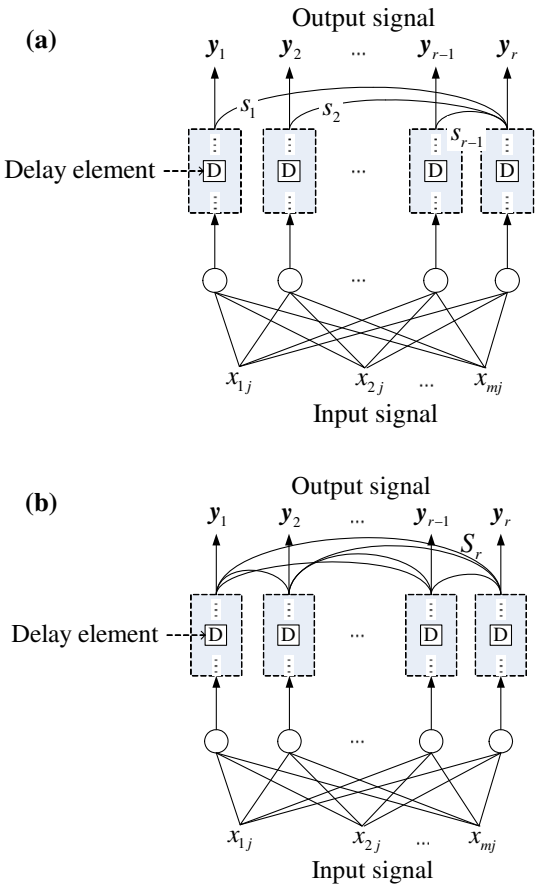

Fig. 2 Neuron model for estimating several principal components adaptively. **a** Unparallel model, **b** parallel model. When there is no updating in this neural network, the output is a matrix $[\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_r]^{\mathrm{T}}$ for an image (matrix), where the vector $\boldsymbol{y}_k$ for $k = 1, \ldots, r$ corresponds to the output in Fig. 1

Assume that the weight vectors of the first $r-1$ output neuron have converged to the eigenvectors of $\boldsymbol{R}$ corresponding to the largest $r-1$ eigenvalues. The weight vector of the rth neuron can converge to the eigenvector of $\boldsymbol{R}$ corresponding to the rth largest eigenvalue, subject to being orthonormal with other $r-1$ eigenvectors through learning of this network.

An image $\boldsymbol{X}=(\boldsymbol{x}_1,\boldsymbol{x}_2,\ldots,\boldsymbol{x}_j,\ldots,\boldsymbol{x}_n)$ is input to the network column by column, where $\boldsymbol{x}_j=(x_{1j},x_{2j},\ldots,x_{mj})^{\mathrm{T}}$. Thus, we obtain an $(r-1)\times n$-dimensional projected matrix $\boldsymbol{Y}=[\boldsymbol{y}_1,\boldsymbol{y}_2,\ldots,\boldsymbol{y}_{r-1}]^{\mathrm{T}}$ from the first $r-1$ neuron output. The feedforward connection weight matrix $\boldsymbol{W}=(\boldsymbol{w}_1,\boldsymbol{w}_2,\cdots,\boldsymbol{w}_{r-1})$ is constructed by the weight vectors of the first $r-1$ neuron. The weight vector of the rth neuron which links the front $m-1$ neurons is $\boldsymbol{s}=(s_1,s_2,\ldots,s_{r-1})$, which is called lateral connection weights.

Accordingly, the relationship between the input and output of the network can be written as

$$\boldsymbol{Y}(t)=\boldsymbol{W}^{\mathrm{T}}(t)\boldsymbol{X}(t) \tag{20}$$

$$\boldsymbol{y}_r(t)=\boldsymbol{w}_r^{\mathrm{T}}(t)\boldsymbol{X}(t)+\boldsymbol{s}(t)\boldsymbol{Y}(t) \tag{21}$$

The feedforward connection weights and the lateral connection weights are updated in accordance with the standard Hebbian learning rule given as

$$\boldsymbol{w}_r(t+1)=\boldsymbol{w}_r(t)+\beta[\boldsymbol{X}(t)\boldsymbol{y}_r^{\mathrm{T}}(t)-\boldsymbol{y}_r(t)\boldsymbol{y}_r^{\mathrm{T}}(t)\boldsymbol{w}_r(t)] \tag{22}$$

$$\boldsymbol{s}(t+1)=\boldsymbol{s}(t)+\gamma[\boldsymbol{Y}(t)\boldsymbol{y}_r^{\mathrm{T}}(t)-\boldsymbol{y}_r(t)\boldsymbol{y}_r^{\mathrm{T}}(t)\boldsymbol{s}(t)] \tag{23}$$

3.2 Convergence discussion

Assume that the weight vectors $\boldsymbol{w}_1(t),\boldsymbol{w}_2(t),\ldots,\boldsymbol{w}_{r-1}(t)$ of the first $r-1$ neurons have converged, respectively, the eigenvectors $\boldsymbol{\varphi}_1,\boldsymbol{\varphi}_2,\ldots,\boldsymbol{\varphi}_{r-1}$ of $\boldsymbol{R}$ corresponding to the largest $r-1$ eigenvalues, that is,

$$\boldsymbol{W}(t)=(\boldsymbol{\varphi}_1,\boldsymbol{\varphi}_2,\cdots,\boldsymbol{\varphi}_{r-1}) \tag{24}$$

$\boldsymbol{w}_r(t)$ can be represented by the following linear equation

$$\boldsymbol{w}_r(t)=\sum_{i=1}^{n}\theta_i(t)\boldsymbol{\varphi}_i \tag{25}$$

From Eqs. (20) and (21), we may rewrite Eq. (22) as

$$\begin{aligned}\boldsymbol{w}_r(t+1)=\boldsymbol{w}_r(t)+\beta[&\boldsymbol{w}_r(t)\boldsymbol{X}(t)\boldsymbol{X}^{\mathrm{T}}(t)\\&+\boldsymbol{s}(t)\boldsymbol{W}(t)\boldsymbol{X}(t)\boldsymbol{X}^{\mathrm{T}}(t)-\boldsymbol{y}_r(t)\boldsymbol{y}_r^{\mathrm{T}}(t)\boldsymbol{w}_r(t)]\end{aligned} \tag{26}$$

Therefore, the statistical average of $\boldsymbol{w}_r(t+1)$ can be written as

$$\begin{aligned}\boldsymbol{w}_r(t+1)&=\boldsymbol{w}_r(t)+\beta[\boldsymbol{w}_r(t)\boldsymbol{R}+\boldsymbol{s}(t)\boldsymbol{W}(t)\boldsymbol{R}-\mathrm{E}[\boldsymbol{y}_r(t)\boldsymbol{y}_r^{\mathrm{T}}(t)]\boldsymbol{w}_r(t)]\\&=\boldsymbol{w}_r(t)+\beta[(\boldsymbol{w}_r(t)+\boldsymbol{s}(t)\boldsymbol{W}(t))\boldsymbol{R}-\sigma(t)\boldsymbol{w}_r(t)]\end{aligned} \tag{27}$$

where $\sigma(t)=\mathrm{E}[\boldsymbol{y}_r(t)\boldsymbol{y}_r^{\mathrm{T}}(t)]$, and $\boldsymbol{R}=\mathrm{E}[\boldsymbol{X}(t)\boldsymbol{X}^{\mathrm{T}}(t)]$.

The learning rule of θ_i can also be written according to Eqs. (25) and (27)

$$\begin{aligned}\theta_i(t+1)&=\theta_i(t)+\beta\lambda_i\theta_i(t)+\beta s_i(t)\lambda_i\theta_i(t)-\beta\sigma(t)\theta_i(t)\\&=[1+\beta(\lambda_i-\sigma(t))]\theta_i(t)+\beta\lambda_i s_i(t)\end{aligned} \tag{28}$$

where λ_i is the ith eigenvalue of $\boldsymbol{R}$.

Similarly, the statistical average in Eq. (23) can be written as

$$s_i(t+1)=\gamma\lambda_i\theta_i(t)+[1+\gamma(\lambda_i-\sigma(t))]s_i(t) \tag{29}$$

Equations (28) and (29) can be written as follows

$$\begin{bmatrix}\theta_i(t+1)\\ s_i(t+1)\end{bmatrix}=\begin{bmatrix}1+\beta(\lambda_i-\sigma(t)) & \beta\lambda_i\\ \gamma\lambda_i & 1+\gamma(\lambda_i-\sigma(t))\end{bmatrix}\begin{bmatrix}\theta_i(t)\\ s_i(t)\end{bmatrix} \tag{30}$$

when $i\geq r$, $s_i(t)=0$. We rewrite $\theta_i(t+1)$ in Eq. (29)

$$\theta_i(t+1)=[1+\beta(\lambda_i-\sigma(t))]\theta_i(t) \tag{31}$$

Rewrite $\sigma(t)$:

$$\begin{aligned}\sigma(t)&=\mathrm{E}[\boldsymbol{y}_r(t)\boldsymbol{y}_r^{\mathrm{T}}(t)]\\&=\mathrm{E}[\boldsymbol{w}_r(t)\boldsymbol{X}(t)\boldsymbol{X}^{\mathrm{T}}(t)\boldsymbol{w}_r^{\mathrm{T}}(t)]\\&=\boldsymbol{w}_r(t)\boldsymbol{R}\boldsymbol{w}_r^{\mathrm{T}}(t)\\&=\left(\sum_{i=1}^{n}\theta_i(t)\boldsymbol{\varphi}_i\right)R\left(\sum_{i=1}^{n}\theta_i(t)\boldsymbol{\varphi}_i^{\mathrm{T}}\right)\\&=\sum_{i=1}^{n}\lambda_i\theta_i^2(t)\end{aligned} \tag{32}$$

When $t\to\infty$,

$$\sigma(t)=\sum_{i=m}^{n}\lambda_i\theta_i^2(t) \tag{33}$$

Suppose $\theta_r(t)\neq 0$, and let

$$\alpha_i(t)=\frac{\theta_i(t)}{\theta_r(t)},\quad r=r+1,r+2,\ldots,n \tag{34}$$

From Eq. (31), we can obtain

$$\alpha_i(t+1)=\frac{1+\beta(\lambda_i-\sigma(t))}{1+\beta(\lambda_r-\sigma(t))}\alpha_i(t) \tag{35}$$

Because of the eigenvalues $\lambda_1>\lambda_2>\cdots>\lambda_r>\cdots>\lambda_n>0$ of R,

$$\frac{1+\beta(\lambda_i-\sigma(t))}{1+\beta(\lambda_r-\sigma(t))}<1 \tag{36}$$

Thus,

$$\lim_{t\to\infty}\alpha_i(t)=0,\quad i=m+1,m+2,\ldots,n \tag{37}$$

As $\theta_r(t)$ has boundary,

$$\lim_{t\to\infty}\theta_i(t)=0,\quad i=m+1,m+2,\ldots,n \tag{38}$$

Thus, Eq. (33) is transformed into

$$\sigma=\lambda_r\theta_r^2(t) \tag{39}$$

Substitute the preceding equation into Eq. (31),

$$\theta_r(t+1)=[1+\beta\lambda_r(1-\theta_r^2(t))]\theta_r(t) \tag{40}$$

From Eq. (40), we can see that

$$\lim_{t\to\infty}\theta_r(t)=1 \tag{41}$$

Therefore,

$$\lim_{t\to\infty}w_r(t)=\boldsymbol{\varphi}_r \tag{42}$$

So when the weight vectors of the $r-1$ neurons converge to the eigenvectors of $\boldsymbol{R}$ corresponding to the first $r-1$ largest eigenvalues $\boldsymbol{\varphi}_1,\boldsymbol{\varphi}_2,\ldots,\boldsymbol{\varphi}_{r-1}$, the weight vector of the rth neuron $w_r(t)$ will converge to the eigenvector of $\boldsymbol{R}$ corresponding to the rth eigenvalue $\boldsymbol{\varphi}_r$. Particularly, when $r=1$, the aforementioned algorithm will be just an estimation of the first principal component, and its weight vector will converge to the eigenvector of $\boldsymbol{R}$ corresponding to the largest eigenvalue.

3.3 Components estimation learning steps

Equations (22) and (23) are the kernel of the components estimation learning algorithm. The feedforward and lateral connection weights should be updated according to Eqs. (22) and (23). Therefore, the stepwise process proceeds as follows:

Step (1): Set $r=1$ and pre-assign the number of neurons.

Step (2): Initialize $\boldsymbol{w}_r(0)$ to some random values and initialize $\boldsymbol{s}(0)$ to an all-zero matrix;

Step (3): Select the learning rate parameters β andγ;

Step (4): Compute the update for the feedforward connection weights according to Eq. (22) and compute the update for the lateral connection weights according to Eq. (23);

Step (5): Compute the errors $\|\boldsymbol{w}_r(t+1)-\boldsymbol{w}_r(t)\|_F$ and $\|\boldsymbol{s}(t+1)-\boldsymbol{s}(t)\|_F$, where $\|\|_F$ denotes the Frobenius norm, and if either of the errors is larger than the set value, then go to the step (4); else, $r=r+1$, if $r<p$ (p denotes the number of principal components needed), then go to the step (2), otherwise stop.

3.4 Parallel version

Since each neuron should be added after the convergence of the previous ones in the model shown in Fig. 2a, each node does not get affected by any nodes following it. However, perhaps the simplest way to implement the adaptive estimation of several principal components is to follow the parallel version that will extract the principal components in parallel rather than one after the other. The first component can be extracted by the model shown in Fig. 1; therefore, the first component is unaffected by other nodes since it has no prior nodes. And the second neuron can begin converging to the second component no later than the first one converges. Similarly, the rth neuron can begin converging to the rth component no later than the $(r-1)$th neuron has converged. The network for the parallel version is shown in Fig. 2b. The stepwise process proceeds for all neurons in parallel as follows: (1) Initialize $\boldsymbol{w}_r(0)$ to some random values; initialize $\boldsymbol{s}(0)$ to an all-zero matrix; pre-assign the number of neurons. (2) Select the learning rate parameters β and γ; (3) Update for the feedforward connection weights and the lateral connection weights according to Eqs. (22) and (23), until the stopping criterion is satisfied, that is, the Frobenius norm of the difference between weight vectors of two consecutive iterations is little enough.

The space complexity of this proposed 2DPCA NN implementation is $O(n)$ in Eqs. (22) and (23) for the step (3), which is much inferior to $O(n^3)$—the complexity of standard 2DPCA based on statistics using Jacobian method. The Jacobian method's space complexity will be analyzed together with time complexity in Sect. 4.6.

3.5 Convergence property test

We selected a set of close data with the size of 16×512. Each datum sample is represented as a vector, and a collection of data is represented as a single large matrix, where each row of the data matrix corresponds to a data point and each column corresponds to a feature. These data, which mean 512 samples of dimension 16, are used to test the iteration of original NNs adaptive estimation of PCA (called 1D NN for short). From the view of computation model of the proposed NN, we reshaped these data into $16\times16\times32$. Under this new model, each datum is a matrix with the size of 16×32, and the collection of data is represented as 16 matrices. When testing with the proposed NN, we import the first, second, third, etc., column of the first matrix and then do the same thing to the second, third, etc., matrix.

For our algorithm, the approximate average error ($AAR(t)$) of the t iterations is defined as

$$AAR(t)=\frac{1}{t}\sum_{i,j}\left\|\boldsymbol{A}_j-\boldsymbol{W}(i)[\boldsymbol{W}^{\mathrm{T}}(i)\boldsymbol{A}_j]\right\|_F^2 \tag{43}$$

where t denotes the total number of iterations; $\boldsymbol{W}(i)$ is a feedforward connection weight matrix for the ith iteration;

$\boldsymbol{A}_j$ and $\boldsymbol{W}(i)[\boldsymbol{W}^{\mathrm{T}}(i)\boldsymbol{A}_j]$ are the jth original image and its estimated image. Similarly, the $AAR(t)$ for 1D NN can be formulated in the following way

$$AAR(t) = \frac{1}{t}\sum_{i=1}^{t} \left\|\mathrm{X} - \boldsymbol{W}(i)[\boldsymbol{W}^{\mathrm{T}}(i)\mathrm{X}]\right\|_F^2 \tag{44}$$

where X is the set of input vectors, and each column of X is one sample.

The AAR brought in by the proposed adaptive neural networks formulation for the 2DPCA with the increasing iteration times is plotted in Fig. 3a which shows that the AAR decreases as the number of feedforward connection weights r increases at rate $= 0.01$. This rate refers to the learning rate β. The observation can be summarized that the plots of AAR for each variable r are all very low at the first iteration, then quickly rise and finally stabilize after a certain number of iterations. It is because $\boldsymbol{w}_r(0)$ is initialized to some random values before iterations and its corresponding AAR is also some random value; and system response requires a process, but $\boldsymbol{W}^{\mathrm{T}}(i)$ is changed toward the true value with its standard learning rule. Finally, the low AAR after several iterations never improves upon the result of the convergence of this NN. This NN almost converges after 80 iterations. The higher the number of feedforward connection weights is, the lower the AAR is. Similarly, Fig. 3b shows the convergence of AAR at different rates {0.0001, 0.001, 0.01, 0.02, 0.05} with the increasing iteration times when the number of feedforward connection weights is equal to 5. Here, the suitable values the learning rate parameters β and γ are 0.05 and 5 through experiments. The conclusion drawn here is that the proposed NN achieves a shorter time of iteration when a suitable value of the learning rate is chosen.

Figure 4 shows the fluctuations and slow convergence of AAR of original neural networks adaptive estimation of PCA as iteration gradually increases. Figure 4a, b corresponds to Fig. 3a, b, respectively, and they have the same parameters. It is found that this NN needs a larger number of iterations for convergence, but achieves a lower AAR than the proposed NN. The value of AAR is related to the number of samples and the distribution of the eigenvalue. Generally speaking, the larger the proportion of the sum of the eigenvalues corresponding to the selected eigenvectors to the summation of all eigenvalues is, the lower the AAR is, vice versa. In addition, the number of samples exerts some influence on the distribution of eigenvalue. The detailed discussion about the error problem is given in Sect. 4.3, and we will divide the instances of sample volume into two different cases, i.e., a large-sample test and a small-sample test.

Comparisons between the local enlarged plot of Fig. 4b and the AAR under that the learning rate is equal to 0.05 of

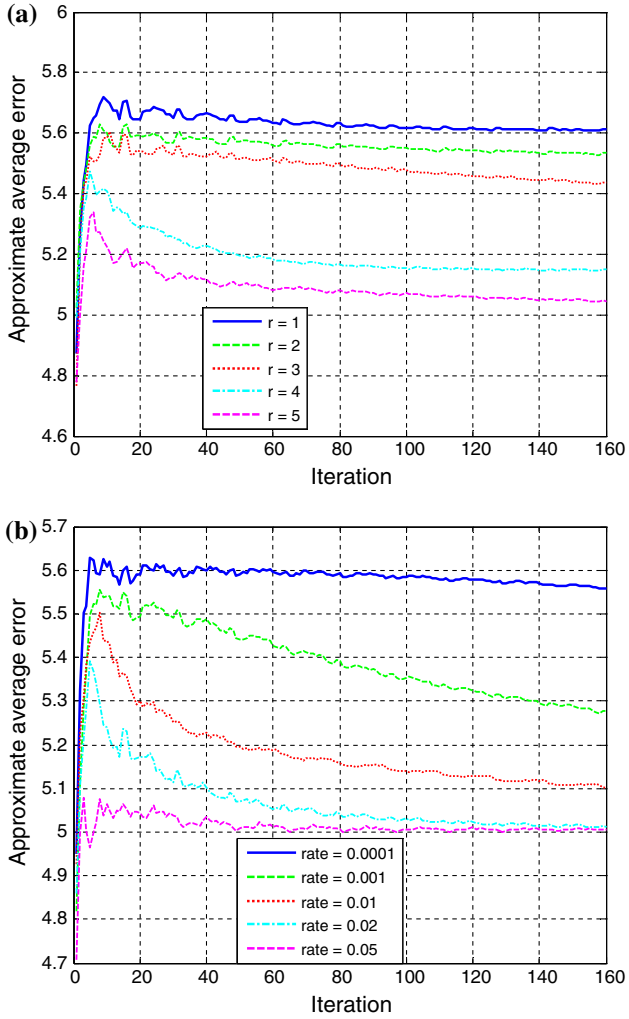

Fig. 3 Approximate average error of the proposed NN. **a** Under different values of r (rate $= 0.01$); **b** under different values of rate ($r = 5$)

the proposed NN are shown in Fig. 5. It reveals that the AAR of 1D NN fluctuates acutely under different parameters during the range [1160] of iteration, and the proposed NN is superior to the 1D NN when a small number of iterations are concerned.

4 Performance analysis and simulation results

In this section, computer simulations are conducted to assess the performance of the proposed adaptive neural networks formulation for the 2DPCA in support of the following three objectives:

1. Investigation of the properties of eigenfaces and eigengaits computed by the proposed adaptive neural networks formulation algorithm;

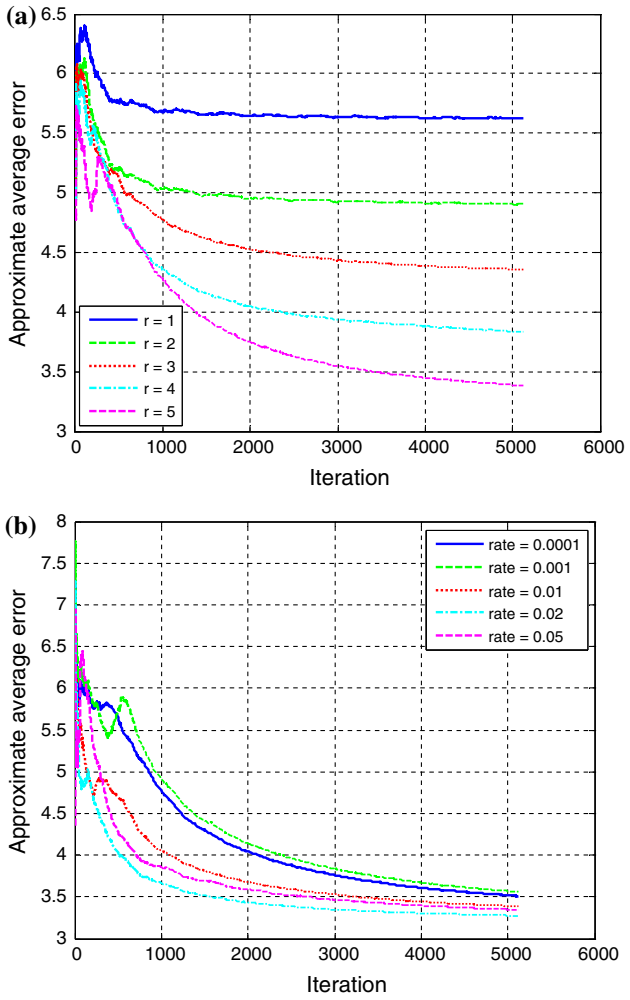

Fig. 4 Approximate average error of neural networks adaptive estimation of PCA. **a** Under different values of r (rate = 0.01); **b** under different values of rate ($r = 5$)

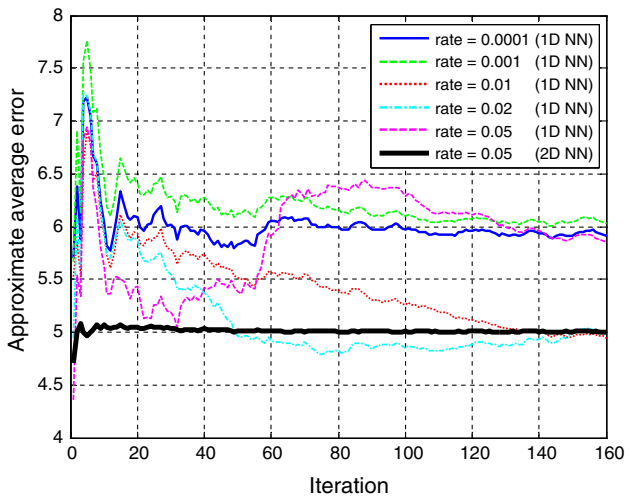

Fig. 5 Comparisons of the proposed NN (2D NN) and 1D NN

2. Evaluation of the proposed adaptive neural networks formulation on such problems as reconstruction error, generalization capability and classification.
3. Comparison between the proposed adaptive neural networks formulation and its statistical version.

Before presenting the experimental results, the experimental data are described first.

4.1 Experimental data

Dataset A AR [39] is a well-known database for face recognition. It contains 120 persons participated in different facial expressions and variations over time, for a total of 1680 cropped images of 50 × 40 pixels. The images for one person are shown in Fig. 6a.

Dataset B CASIA(B) gait database [40] includes a total of 124 persons, and each person has 10 sequences which are six normal gaits, two gaits with a bag and two gaits with a coat. We choose the normal ones who walk on a straight-line path at natural cadences in a viewing angle with respect to the image plane, namely a 90 degree as the evaluation samples. We use a dual-ellipse fitting approach for robust gait periodicity detection [41]. The gait energy images (GEIs) have already been extracted as the gait characteristic for each gait sequence by us [4]. In order to eliminate the influence of the image size on performance accuracy, the size of all images has been unified into 64 × 64 pixels with each silhouette centralized as in Fig. 6b.

4.2 Eigenfaces for weight vectors

PCA has generated a set of eigenfaces by performing a mathematical process on a large set of images depicting different human faces. These eigenfaces can be considered as a set of standardized face ingredients derived from statistical analysis of many pictures of faces. Any human face can be regarded as a combination of these standard faces. Every face image can be projected into the subspace spanned by all the eigenvectors. Therefore, each face image corresponds to a point in the subspace. Likewise, every point in the subspace also denotes a certain image in correspondence. Eigenfaces obtained from a neural network adaptive estimation algorithm of PCA are shown in Fig. 7a.

Furthermore, the proposed NN is applied to solve 2D eigenface problem. 2D eigenface is expressed like a facial image. Denote n to be the number of columns in an image, and then, an outline of the 2D eigenface procedure can be illustrated as follows.

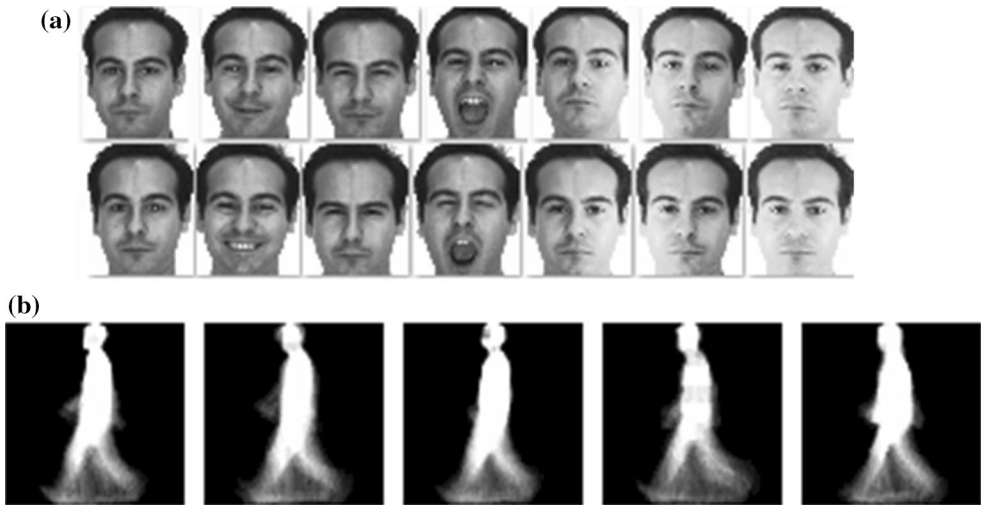

Fig. 6 **a** Face images for one subject from AR database. **b** GEIs of different gait sequences

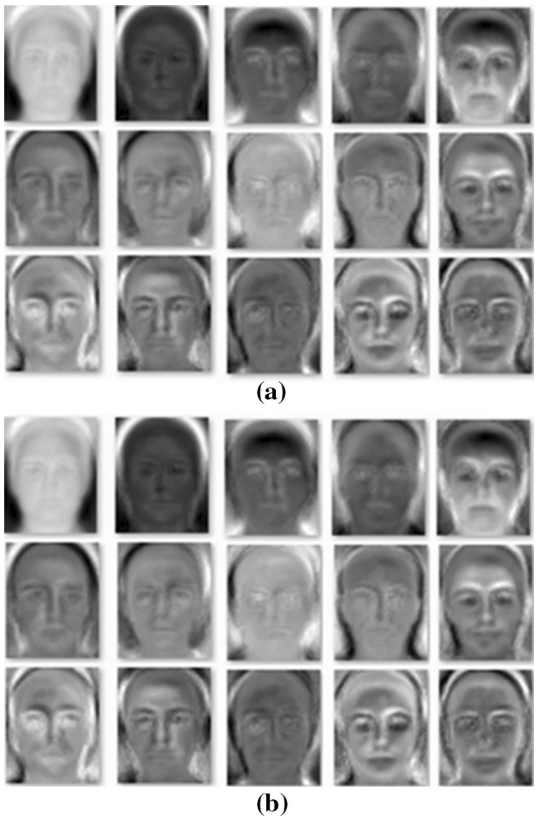

Fig. 7 Eigenfaces. **a** From neural network adaptive estimation algorithm of PCA; **b** the proposed NN

for i=1:n

t is a spliced matrix by putting transposed i-th column of all the samples from up to down

base$\{i\} = t^{\mathrm{T}} * [\, \boldsymbol{w}_1(t)\,,\ \boldsymbol{w}_2(t)\,, \cdots, \boldsymbol{w}_{15}(t)\,]$

end

for k=1:15

for j=1:n

2DEigenface(k) =[base{1}(:,k),⋯,base{j}(:,k),⋯, base{n}(:,k)];

end

save 2DEigenface(k)

end

The weight vectors $\boldsymbol{w}_1(t)$, $\boldsymbol{w}_2(t)$, …, $\boldsymbol{w}_{15}(t)$, whose corresponding eigenfaces are shown in Fig. 7b, are the first 15 feedforward connection weights obtained from this NN. Compared with the results in Fig. 7a, it can be inferred that both eigenfaces for the neural networks formulation algorithm of 1D**PCA** and 2DPCA are the same. The conceptions of eigenfaces in both 1D**PCA** and 2DPCA NNs are uniform. This pattern of eigenfaces is how different features of a face are singled out to be evaluated and scored.

In the eigengaits experiments, as shown in Fig. 8a, b, respectively, neural networks of 1DPCA and 2DPCA can

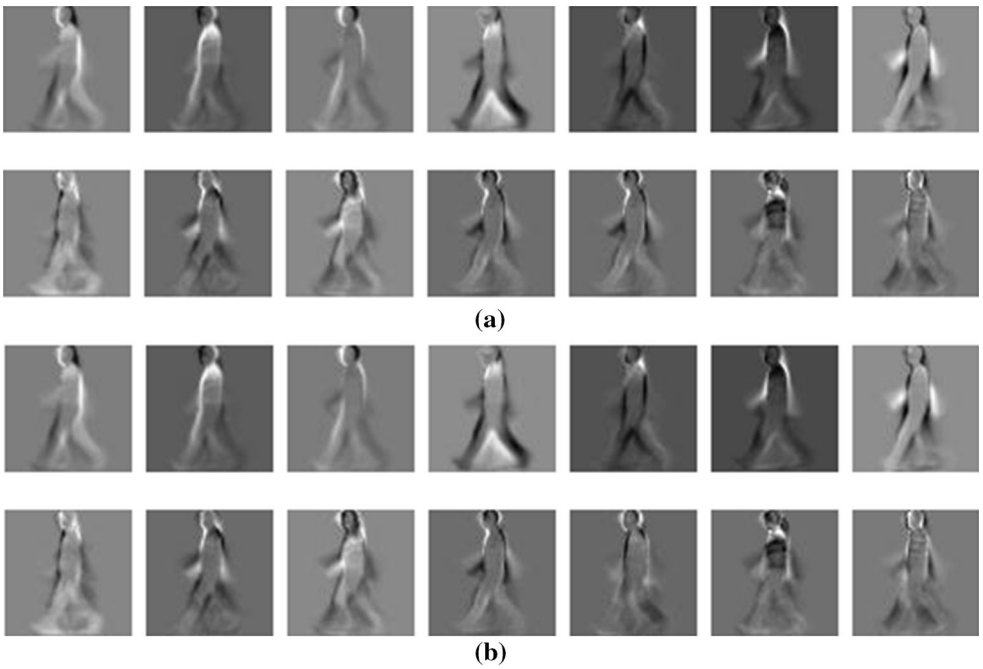

Fig. 8 Eigengaits. **a** From neural network adaptive estimation algorithm of PCA; **b** the proposed NN

also get the same specific pattern. Although 1D and 2D principal components are computed by PCA and 2DPCA NNs separately, their eigenfaces (or eigengaits) are uniform.

4.3 Reconstruction error discussion

This subsection aims to compare the reconstruction error of neural networks of 1DPCA with 2DPCA under the condition of equal number of dominant principal components. This implies approximately the same computational cost. Most researchers usually only focus on a large-sample test and ignore a case of a small sample. However, it is necessary that the instances of sample volume are separated into two species which will be widely divergent, namely large sample and small sample.

4.3.1 Large-sample test

In our experiments, we use the whole database, so the dataset A has 1680 samples and the dataset B has 744 samples totally.

4.3.1.1 Dataset A experiments Carried out by simply performing an 'inverse vec' operation of Eq. (20), the image must be reconstructed. See Fig. 9a for an example of the reconstructed face image—from the former to the latter sub-image—using first 5, 10, …, 40 feedforward connection weights for the first person. Comparison among the eight reconstructed face images in Fig. 9a reveals a very

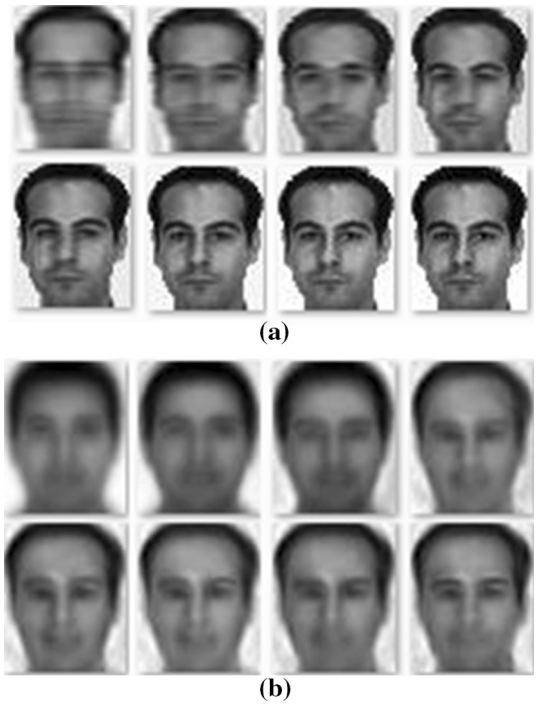

Fig. 9 Reconstruction. **a** The proposed NN; **b** neural network adaptive estimation algorithm of PCA

distinct image for different numbers of feedforward connection weights. The neural network architecture with 25-neuron can reconstruct a clear and distinguishable face image. In contrast, Fig. 9b shows eight reconstructed images from an equal number of feedforward connection

weights using a neural network adaptive estimation algorithm of PCA, from which we can see that these reconstructed results are far more blurred than the proposed NN. Figure 10a shows the reconstruction errors over the variation of the number r of feedforward connection weights for the two above-mentioned NNs. The reconstruction error here is computed by root mean square error (RMSE).

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\|A_i - \hat{A}_i\right\|_F^2} \tag{45}$$

where A_i and $\hat{A}_i$ $(i = 1, 2, \ldots, n)$ are an original and reconstructed images. n denotes the total number of samples.

4.3.1.2 Dataset B experiments The reconstruction results of neural networks of 2DPCA and 1DPCA are shown in Fig. 11a, b, respectively. Similarly, the first sub-images in the two figures are corresponding to the original GEI, and the second to the last sub-images in them are the reconstructed GEIs by the 10, 20, …, 60 feedforward connection weights. Since it is difficult for us to tell the difference between them, we have evaluated the problem of the reconstruction error as the number of feedforward connection weights increases gradually, which is shown in Fig. 10b.

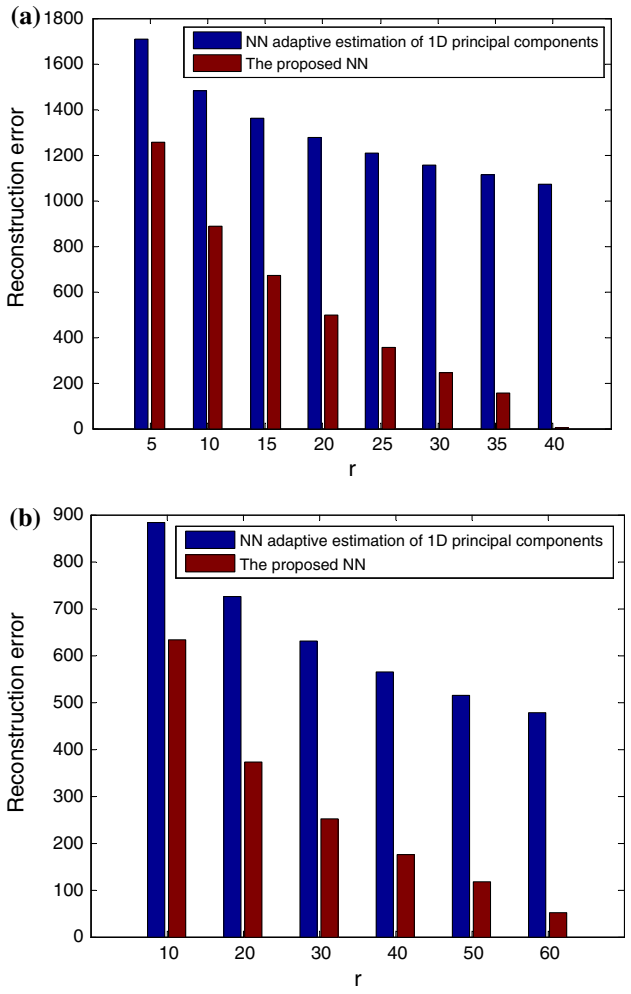

Fig. 10 Reconstruction error. **a** In the whole dataset A; **b** in the whole dataset B

From Fig. 10, we can have the following observation: The adaptive neural networks formulation for the 2DPCA achieves a lower residue error than that for the 1DPCA when the training sample set is large.

4.3.2 Small-sample test

The reconstruction error problem is examined on the small sample only from a single person; namely, the numbers of samples selected from the dataset A and B are 14 and 6, respectively.

4.3.2.1 Dataset A experiments We have also tested two neural networks for both 2DPCA and 1DPCA, and the reconstruction results are shown in Fig. 12a, b. They correspond to the face images reconstructed by the 1, 2, …, 8 feedforward connection weights (i.e., feature dimension). We can clearly see blurred results of the proposed method, but it is inferior to the reconstruction results of NN for PCA, whose neural network architecture with 5-neuron can reconstruct a clear and distinguishable face image. The reason for this is twofold: firstly, the top eigenvectors in NN for PCA reflect the reconstruction information; and secondly, there are 13 eigenvectors in total, the top 5 of which have occupied majority energy. In contrary, the number of eigenvectors in the proposed NN is large, and it is not enough to select a few eigenvectors to reconstruct human faces. Figure 13a displays the reconstruction errors of using these two NNs for a single person's samples from the dataset A.

4.3.2.2 Dataset B experiments The reconstruction results of neural network algorithms of 2DPCA and 1DPCA are shown in Fig. 14a, b, respectively. Similarly, the first sub-images in the two figures are corresponding to the original GEIs, and the second to the last sub-images in them are the reconstructed GEIs by the 1, 2, …, 5 feedforward connection weights. Figure 13b displays the reconstruction errors of using these two NNs for a single person's samples from the dataset B.

From the results of reconstruction errors in Fig. 13, it should be pointed out that the adaptive neural networks formulation for the 2DPCA achieves a higher residue error than that for the 1DPCA, when the training sample set is small.

Comparisons between Case 1 and Case 2.

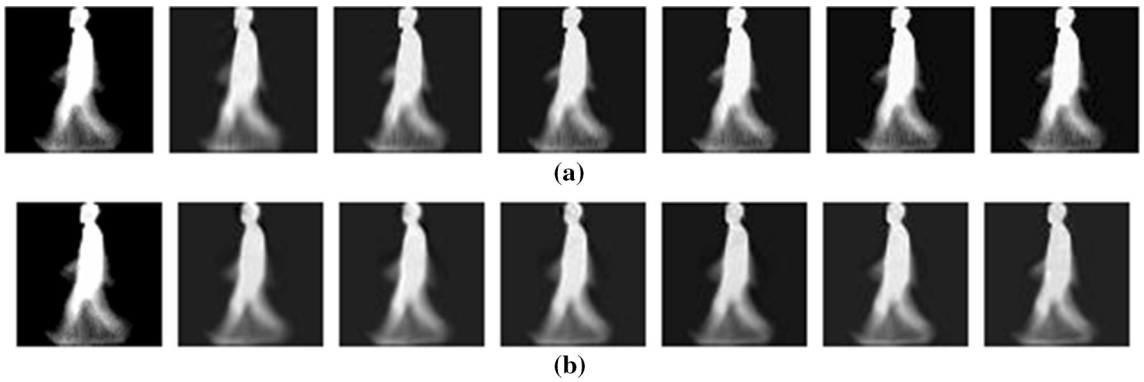

Fig. 11 Reconstruction. **a** The proposed NN; **b** neural network adaptive estimation algorithm of PCA

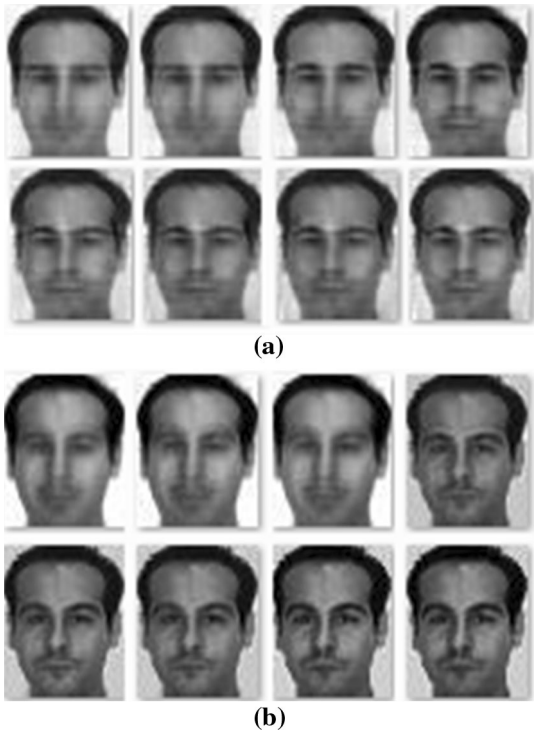

Fig. 12 Reconstruction. **a** The proposed NN; **b** neural network adaptive estimation algorithm of PCA

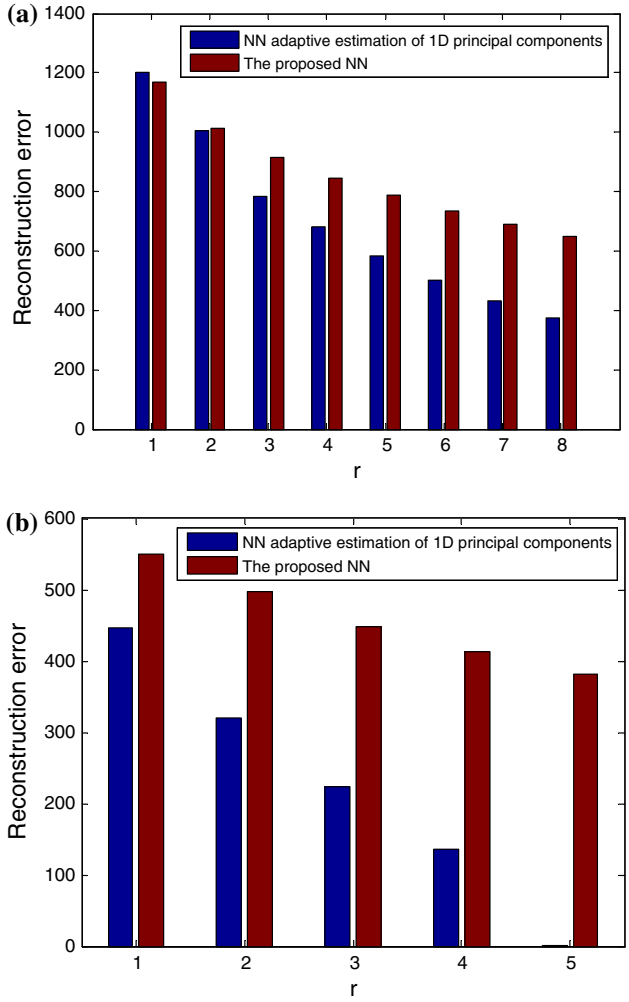

Fig. 13 Reconstruction error. **a** A single person's samples from dataset A; **b** a single person's samples from dataset B

Obviously, the above experiments for large sample and small sample differ significantly. We can take the dataset B for example, and there are 744 and 6 samples in total with the size of 64 × 64 pixels for large and small sample set, respectively.

For a large sample set, 1DPCA has 743 nonzero eigenvalues at most, while 2DPCA has 64 ones at the maximum. If we select the same number of feedforward connection weights, which correspond to different eigenvalues, 1DPCA achieves a smaller proportion of the sum of the eigenvalues corresponding to the selected eigenvectors to the summation of all eigenvalues which means a lower energy. On the contrary, 2DPCA gains a larger proportion and a higher energy. Therefore, 2DPCA attains a lower error than 1DPCA.

Conversely, for a small sample set, 1DPCA has five nonzero eigenvalues at most, while 2DPCA has 64 at the maximum. When we also pick out the same number of

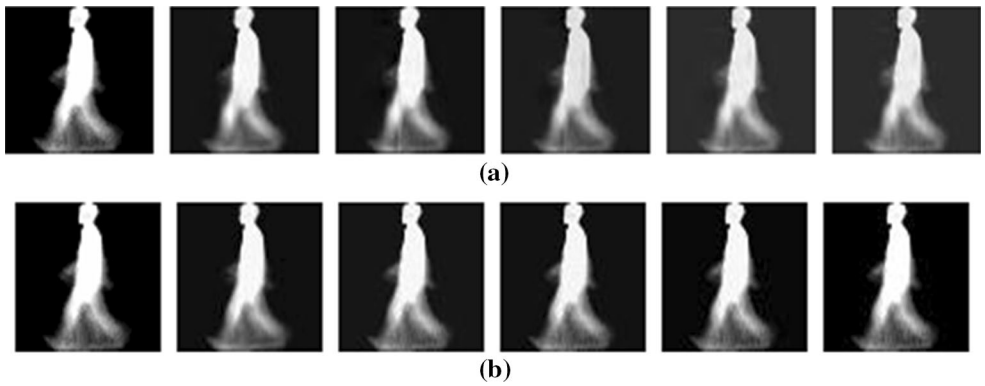

Fig. 14 Reconstruction. **a** The proposed NN; **b** neural network adaptive estimation algorithm of PCA

Fig. 15 Reconstruction for the proposed neural network

feedforward connection weights, 1DPCA comes through a larger proportion of the sum of the eigenvalues corresponding to the selected eigenvectors to the summation of all eigenvalues, in other words a higher energy than 2DPCA. As a result, 2DPCA acquires a larger error than 1DPCA. Although the performance of 2DPCA is expected to be better with the augmentation of dimensionality, the complexity will rise. The motivation of this subsection is to test reconstruction error ensuring approximate complexity in the utmost small-sample situation.

4.4 Generalization capability discussion

To illustrate the generalization capability of the proposed NN, we test the reconstruction of another face image which is collected from the Internet. The results are shown in Fig. 15, where the first sub-image shows the original girl face image and others are reconstructed faces coded using the subspace learning rule with 5, 10, …, 40 feedforward connection weights, respectively, just like the experiment in the Sect. 4.3-Case 1-(1). Apparently, the girl face image and the man's face image in Fig. 9a are statistically similar because of the relatively good coding performance that is evident in Fig. 15 when the whole dataset A is used to obtain the feedforward connection weights. The more the number of feedforward connection weights is, the better the reconstruction result will be. However, a high number of feedforward connection weights is still the number one cause of computational effort. It is clear that the reconstructed image has a good visual quality when using only the top 25 feedforward connection weights.

4.5 Classification

Besides the experiments on reconstruction, we will compare the proposed NN with the NN of 1DPCA on classification. It does not make sense to conduct experiments in a small sample set. Accordingly, several experiments for a large sample set are carried out to show the effectiveness of the proposed NN for face and gait recognition.

For each person in the dataset A, for example, we use the first column in Fig. 6a as two samples for training and the rest of eight samples (as shown in the second to the fourth column in Fig. 6a) with varying facial expressions for testing. The first half samples of each person are used for training, and the remainders are used for testing for the dataset B. The nearest neighbor classifier is employed for classification, and the recognition performance is measured in accuracy.

4.5.1 Dataset A experiments

The accuracy of the proposed NN and NN adaptive estimation of 1DPCA over the variation of number of the feedforward connection weights is plotted in Fig. 16a. The proposed NN achieves its maximal accuracy of 95.83 % using only 10 feedforward connection weights. In addition, Fig. 16a shows the proposed NN consistently outperforming NN adaptive estimation of 1DPCA irrespective of the variation in number of weights.

4.5.2 Dataset B experiments

Figure 16b also shows a plot of accuracy versus the number of the feedforward connection weights. As can be

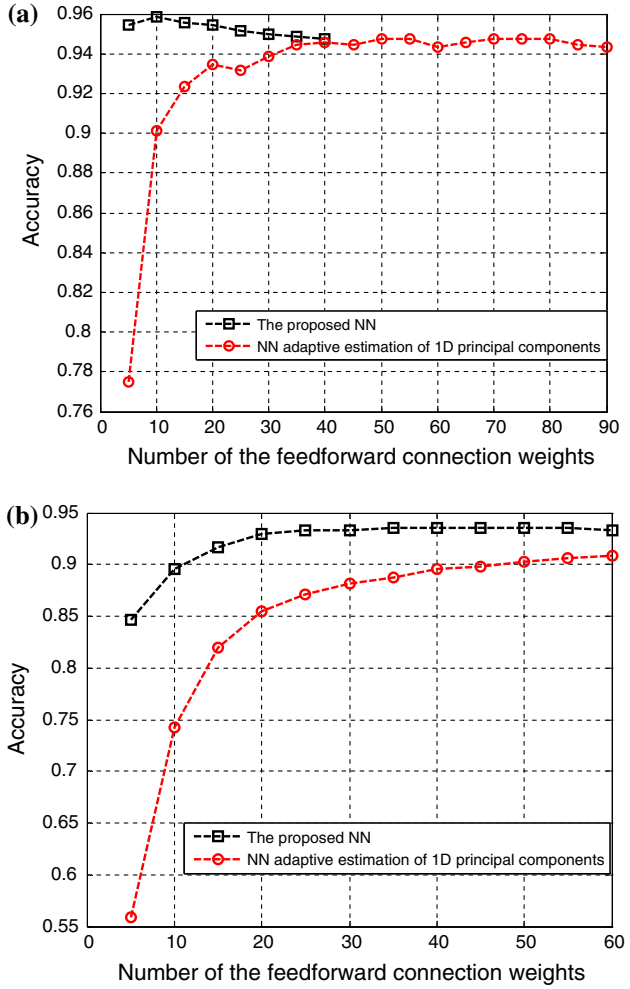

Fig. 16 Accuracy. **a** Dataset A; **b** dataset B

seen, the proposed NN achieves 93.55 % accuracy with 35 feedforward connection weights, compared with 90.86 % accuracy with 60 feedforward connection weights. For NN adaptive estimation of 1DPCA, when more than 120 feedforward connection weights are used to constitute the network, the accuracy then improves only slightly. Even if the number of feedforward connection weights increases to 190, this NN reaches a 93.55 % accuracy rate.

The above comparative evaluations demonstrate that the proposed NN is more effective with fewer feedforward connection weights, which suggests less computational complexity than NN adaptive estimation of 1DPCA.

4.5.3 Discussion

The reason why the experiment validation considers cases where reduced dimension is limited to a few is that much empirical work has been done to determine how many components should be computed to adequately represent data. Generally, an appropriate value for components is desirable as small as possible while achieving a reasonably high value on a percentage basis. For example, the cumulative energy is higher than a certain threshold, say 90 %. Eigenvalues and eigenvectors always appear in pairs, and the feedforward connection weights (i.e., eigenvectors corresponding to eigenvalues) are automatically arranged in a descending order. It turns out that the eigenvector with the largest eigenvalue represents the principle component. The significant information mainly centralizes a few components. The components of less significance can be ignored. Some information may be definitely lost but not that much if the eigenvalues can be small.

4.6 Comparison with statistical 2DPCA method

Suppose we need to calculate the eigenvectors for a $n \times n$ Matrix A with our proposed neutral network, the Jacobian method can be broken into five steps: First, a nonzero and non-diagonal element with max absolute value a_{ij} is selected; for this step, the time complexity is $O(1)$; second, we calculate θ from $\tan 2\theta = \frac{2a_{ij}}{a_{ii}-a_{ij}}$; here, we look up the inverse trigonometric table for θ value, and thus, planar rotation matrix P_1 can be get. During this process, if n is not relatively big, the time complexity for computing the θ and the subsequent Matrix A_1 can be considered as $O(1)$; third, we calculate each element in Matrix A_1 from $A_1 = P_1^{\mathrm{T}} A P_1$, due to the fact that P_1 is a sparse matrix with finite ones in diagonal while more zeros in other position, this P_1 can be split into two matrices, and the final amount of calculation is O(n); fourth, we substitute A with A_1, repeat Steps 1, 2 and 3 to calculate A_2 and P_2 and continue this process till the elements in diagonal of A_m are small enough (i.e., smaller than the allowed errors). This step's time complexity is $O(n^2)$; The last step is that diagonal elements of A_m are approximation for all eigenvalues of matrix A, and the jth column of $P = P_1 P_2 \cdots P_m$ is corresponding to the eigenvector of eigenvalue λ_j (the jth element in A_m diagonal). In summary, the total time complexity of this method is $O(n^2)$. Computing P needs $O(n^3)$ space complexity.

In this section, we compare the complexity and performances of our adaptive NN formulation with the statistical 2DPCA method tested on computer with Intel(R) Core(TM)2 Duo T8300@2.40GHZ CPU and 1G RAM. In addition to the superiority of space complexity, the NN formulation has comparative time complexity in terms of the number of elementary operations to the statistical version. Not only does Table 1 provide space complexity, time complexity, but also reconstruction error, accuracy and time-consumed during the whole training and testing processing for both dataset A and B experiments.

Table 1 Complexity and performance comparison

	Adaptive NN formulation	Statistical 2DPCA
Space complexity	$O(n)$	$O(n^3)$
Time complexity	$O(n^2)$	$O(n^2)$
Dataset A		
Reconstruction error	860.02	864.08
Accuracy	95.83 %	95.83 %
Time-consumed (s)	10.3	11.2
Dataset B		
Reconstruction error	50.9	51.1
Accuracy	93.55 %	92.07 %
Time-consumed (s)	15.5	15.9

For the statistical 2DPCA method, the Jacobi algorithm is employed to obtain eigenvalues and eigenvectors. Moreover, it is kept equal for the number of the feedforward connection weights of adaptive NN formulation and the eigenvectors number of the statistical version. It is observed that the adaptive NN formulation method performs quite well, and it can yield comparative accuracy, running time and reconstruction results to the statistical version. The proposed NNs method can be a neuro-computing algorithm to realize 2DPCA, and it possesses very attractive in the potential applications on hardware platforms like single-chip computer and embedded systems to overcome the disadvantages of low cache and memory.

5 Conclusions

In this paper, a new technique for image feature extraction and representation using NN—adaptive neural networks formulation for the 2DPCA—was developed. It was also the first time for the uniform conceptions of 'eigenfaces' in both 1DPCA and 2DPCA neural networks to be put forward. A comparative assessment of the performance of the proposed NN and 1D NN shows that the adaptive neural networks formulation for the 2DPCA achieves a lower residue error than that for the 1DPCA when the training sample set is large. On the contrary, when the training sample set is small, the adaptive neural networks formulation for the 2DPCA achieves a higher residue error than that for the 1DPCA. On face and gait recognition tasks, a simple nearest neighbor classifier test indicated a particular benefit of the neural network developed here serving as an efficient alternative to conventional 1D NN. The proposed NN has a lower computation than a 1D NN on the feature extraction of the same image matrix. Other learning rules for adaptive estimation of neural networks of 2DPCA will be tested in the future work.

Acknowledgments The authors would like to thank Dr. T. Tan from the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, for providing us with the CASIA(B) gait database. This project is supported by the Natural Science Foundation of China (Grant No. 61201370), the Specialized Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20120131120030), the Independent Innovation Foundation for Postdoctoral Scientists of Shandong Province (Grant No. 201303100), the National Science Foundation for Postdoctoral Scientists of China (Grant No. 2013M530321), the Special Program of China Postdoctoral Science Foundation (Grant No. 2014T70636) and the Key Laboratory of Intelligent Perception and Systems for High-Dimensional Information, Ministry of Education (Grant No. 30920140122006). The authors thank the reviewers, Prof. Jian Yang from Nanjing University of Science and Technology, Prof. Yanfeng Gu and Prof. Jiafeng Liu from Harbin Institute of Technology, Prof. Wankou Yang from Southeast University and Prof. Chuanxian Ren from Sun Yat-Sen University for their useful suggestions.

Appendix

Here, we provide a proof of Eq. (29).

$$\begin{aligned}\boldsymbol{s}(t+1) &= \boldsymbol{s}(t) + \gamma[\boldsymbol{y}_r(t)\boldsymbol{Y}^{\mathrm{T}}(t) - \boldsymbol{y}_r(t)\boldsymbol{y}_r^{\mathrm{T}}(t)\boldsymbol{s}(t)] \\ &= s(t) + \gamma[\boldsymbol{w}_r(t)\boldsymbol{X}(t)\boldsymbol{X}^{\mathrm{T}}(t)\boldsymbol{W}^{\mathrm{T}}(t) \\ &\quad + \boldsymbol{s}(t)\boldsymbol{W}(t)\boldsymbol{X}(t)\boldsymbol{X}^{\mathrm{T}}(t)\boldsymbol{W}^{\mathrm{T}}(t) - \boldsymbol{y}_r(t)\boldsymbol{y}_r^{\mathrm{T}}(t)\boldsymbol{s}(t)]\end{aligned}$$

The statistical average of $\boldsymbol{s}(t+1)$ can be written as

$$\begin{aligned}\boldsymbol{s}(t+1) &= \boldsymbol{s}(t) + \gamma[\boldsymbol{w}_r(t)\boldsymbol{R}\boldsymbol{W}^{\mathrm{T}}(t) \\ &\quad + \boldsymbol{s}(t)\boldsymbol{W}(t)\boldsymbol{R}\boldsymbol{W}^{\mathrm{T}}(t) - \sigma(t)\boldsymbol{s}(t)]\end{aligned}$$

Therefore,

$$\begin{aligned}\boldsymbol{s}_i(t+1) &= \boldsymbol{s}_i(t) + \gamma\theta_i(t)\lambda_i + \gamma\boldsymbol{s}_i(t)\lambda_i - \gamma\sigma(t)\boldsymbol{s}_i(t) \\ &= \gamma\boldsymbol{\theta}_i(t)\lambda_i + (1 + \gamma\lambda_i - \gamma\sigma(t))\boldsymbol{s}_i(t)\end{aligned}$$

References

1. Yang J, Zhang D, Alejandro FF, Yang J (2004) Two-dimensional PCA: a new approach to appearance-based face representation and recognition. IEEE Trans Pattern Anal Mach Intell 26(1):131–137
2. Wang L, Wang X, Feng J (2006) On image matrix based feature extraction algorithms. IEEE Trans Syst Man Cybern B Cybern 36:194–197

3. Ren CX, Dai DQ (2010) Incremental learning of bidirectional principal components for face recognition. Pattern Recogn 43(10):318–330
4. Ben X, Wang K, Yan R, POPOOLA Oluwatoyin Pius (2011) Subpattern-based complete two dimensional principal component analysis for gait recognition. Proc Chin Assoc Sci Technol 7(2):16–22
5. Xu A, Jin X, Jiang Y (2006) Complete Two-Dimensional PCA for Face Recognition. In: 18th International conference on pattern recognition, Hong Kong, China, vol 3, pp 481–484
6. Zhang D, Zhou Z (2005) $(2D)^2$PCA: two-directional two-dimensional PCA for efficient face representation and recognition. Neurocomputing 69:224–231
7. Ye J (2004) Generalized low rank approximation of matrices. In: 21st International conference on machine learning, pp 887–894
8. Liu J, Chen S (2006) Non-iterative generalized low rank approximation of matrices. Pattern Recogn Lett 27:1002–1008
9. Lu C, Liu W, An S (2008) A simplified GLRAM algorithm for face recognition. Neurocomputing 72:212–217
10. Kima YG, Songa YJ, Changa UD, Kimb DW, Yunc TS, Ahna JH (2008) Face recognition using a fusion method based on bidirectional 2DPCA. Appl Math Comput 205:601–607
11. Yang J, Liu C (2007) Horizontal and vertical 2DPCA-based discriminant analysis for face verification on a large-scale database. IEEE Trans Inf Forensics Secur 2:781–792
12. Baldi PF, Hornik K (1995) Learning in linear neural networks: a survey. IEEE Trans Neural Netw 6:837–858
13. Diamantaras KI, Kung SY (1996) Principal component neural networks: theory and applications. In: Adaptive and learning systems for signal processing, communications, and control, Wiley, New York
14. Oja E (1982) A simplified neuron model as a principal component analyzer. J Math Biol 15:267–273
15. Oja E, Karhunen J (1985) On stochastic approximation of eigenvectors and eigenvalues of the expectation of a random matrix. J Math Anal Appl 104:69–84
16. Lv JC, Zhang Y, Li YX (2015) Non-divergence of stochastic discrete time algorithms for PCA neural networks. IEEE Trans Neural Netw Learn Syst 26(2):394–399
17. Sanger TD (1989) Optimal unsupervised learning in a single-layer linear feedforward neural network. Neural Netw 2:459–473
18. Oja E (1992) Principal components, minor components and linear neural networks. Neural Netw 5:927–935
19. Kung SY, Diamantaras KI, Taur JS (1994) Adaptive principal component extraction and applications. IEEE Trans Signal Process 42:1202–1217
20. Haykin S (1994) Neural networks—a comprehensive foundation. Macmillan, New York
21. Andreas W, Kurt H (2000) Local PCA Algorithms. IEEE Trans Neural Netw 11(6):1242–1250
22. Kong XY, An QS, Ma HG et al (2012) Convergence analysis of deterministic discrete time system of a unified self-stabilizing algorithm for PCA and MCA. Neural Netw 36:64–72
23. Karhunen J, Oja E, Wang L, Vigario R, Joutsensalo J (1997) A class of neural networks for independent component analysis. IEEE Trans Neural Netw 8(3):486–504
24. Gou S, Jiao L (2005) Image recognition using Synergetic Neural Network. Lect Notes Comput Sci 3497(2):286–291
25. Chen S (1995) Nonlinear time series modeling and prediction using Gaussian RBF networks with enhanced clustering and RLS learning. Electron Lett 31:117–118
26. Tomenko Vladimir (2011) Online dimensionality reduction using competitive learning and Radial Basis Function network. Neural Netw 24(5):501–511
27. Eric CT, James CB, Nikhil RP (1994) Fuzzy Kohonen clustering networks. Pattern Recogn 27(5):757–764
28. Ceylan R, Ozbay Y (2007) Comparison of FCM, PCA and WT techniques for classification ECG arrhythmias using artificial neural network. Expert Syst Appl 33(2):286–295
29. Huang W, Oh SK, Pedrycz W (2014) Design of hybrid radial basis function neural networks (HRBFNNs) realized with the aid of hybridization of fuzzy clustering method (FCM) and polynomial neural networks (PNNs). Neural Netw 60:166–181
30. Alexandridis Antonios K, Zapranis Achilleas D (2013) Wavelet neural networks: a practical guide. Neural Netw 42:1–27
31. Zhang F, Du B, Zhang LP (2015) Saliency-guided unsupervised feature learning for scene classification. IEEE Trans Geosci Remote Sens 53(4):2175–2184
32. Carvajal Gonzalo, Figueroa Miguel (2014) Model, analysis, and evaluation of the effects of analog VLSI arithmetic on linear subspace-based image recognition. Neural Netw 55:72–82
33. Bian W, Tao D (2011) Max-min distance analysis by using sequential SDP relaxation for dimension reduction. IEEE Trans Pattern Anal Mach Intell 33(5):1037–1050
34. Yang WK, Wang ZY, Sun CY (2015) A collaborative representation based projections method for feature extraction. Pattern Recogn 48(1):20–27
35. Yang WK, Sun CY, Zhang L (2011) A multi-manifold discriminant analysis method for image feature extraction. Pattern Recogn 44(8):1649–1657
36. Sun M, Zhao L, Cao W, Xu Y, Dai X, Wang X (2010) Novel hysteretic noisy chaotic neural network for broadcast scheduling problems in packet radio networks. IEEE Trans Neural Netw 21(9):1422–1433
37. Chen W, Jiao L, Li J, Li R (2010) Adaptive NN backstepping output-feedback control for stochastic nonlinear strict-feedback systems with time-varying delays. IEEE Trans Syst Man Cybern B Cybern 40(3):939–950
38. Chang C, Juang J (2008) An adaptive multipath mitigation filter for GNSS applications. EURASIP J Adv Signal Process. http://portal.acm.org/citation.cfm?id=137.6536.1387861
39. Gross R (2005) Face databases. In: Jain AK, Li SZ (eds) Handbook of face recognition, vol 1. Springer, New York, p 22
40. Yu S, Tan D, Tan T (2006) A framework for evaluating the effect of view angle, clothing and carrying condition on gait recognition. In: Proceedings of 18th international conference on pattern recognition, Hong Kong, China, pp 441–444
41. Ben X, Meng W, Yan R (2012) Dual-ellipse fitting approach for robust gait periodicity detection. Neurocomputing 79:173–178