10-13-1998

Extension of the generalized Hebbian algorithm for principal component extraction

Fredric M. Ham

Inho Kim

PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Extension of the generalized Hebbian algorithm for principal component extraction

Fredric M. Ham
Inho Kim

SPIE.

Extension of the Generalized Hebbian Algorithm for Principal Component Extraction

Fredric M. Ham and Inho Kim

Electrical Engineering Program, Florida Institute of Technology, Melbourne, FL 32901-6988

ABSTRACT

Principal component analysis (PCA) plays an important role in various areas. In many applications it is necessary to *adaptively* compute the principal components of the input data. Over the past several years, there have been numerous neural network approaches to adaptively extract principal components for PCA. One of the most popular learning rules for training a single-layer linear network for principal component extraction is Sanger's generalized Hebbian algorithm (GHA). We have extended the GHA (EGHA) by including a positive-definite symmetric weighting matrix in the representation error-cost function that is used to derive the learning rule to train the network. The EGHA presents the opportunity to place different weighting factors on the principal component representation errors. Specifically, if prior knowledge is available pertaining to the variances of each term of the input vector, this statistical information can be incorporated into the weighting matrix. We have shown that by using a *weighted* representation error-cost function, where the weighting matrix is diagonal with the reciprocals of the standard deviations of the input on the diagonal, more accurate results can be obtained using the EGHA over the GHA.

Keywords: principal component analysis, neural network, extended generalized Hebbian algorithm, weighting matrix .

1. INTRODUCTION

1.1 Statement of the Problem

Principal component analysis (PCA)[(1)] has many applications, e.g., data compression and coding (decoding), adaptive beamforming, high resolution spectral analysis (frequency estimation), and pattern recognition, to name a few [1-5, 22, 23, 26]. Many times it is necessary to *adaptively* determine principal components. That is, there may not be enough data available to estimate the covariance matrix of the input data, and then compute the eigenvalues and eigenvectors of the resulting matrix. Instead an adaptive method is needed to compute the principal eigenvectors associated with the inputs, as the data are processed. There have been many neural network architectures and training algorithms developed for adaptive estimation of principal eigenvectors [6-14, 24, 25, 27-32]. One of the most well known is the Generalized Hebbian Algorithm (GHA) for training a single-layer neural network consisting of linear processing elements [17]. The network can estimate as many principal eigenvectors as desired. The standard approach that is used to develop the GHA learning rule involves using a quadratic, representation-error formulation of a cost (energy) function that is minimized over the weight space. Many times the results obtained using the GHA are not accurate enough, therefore, it is desirable to improve the algorithm.

The GHA has been extended (EGHA), i.e., instead of a standard (non-weighted) quadratic representation error cost function, a *weighted* error function is used [13, 21, 22]. That is, the cost function now has a positive-definite symmetric weighting matrix included. The resulting symmetric subspace learning rule now has terms in the gradient expression that can *not* be discarded, as is the case with the GHA. The terms that are discarded in the GHA must be retained in the EGHA unless the weighting matrix in the cost function is the identity matrix. When the weighting matrix in the EGHA is set to the identity matrix, the GHA can be recovered. When the symmetry is *broken* on the appropriate terms leading to the EGHA, more accurate results can be obtained. In particular, a distinct advantage of the EGHA is that *a priori* information can be taken into account that enhances the capability and accuracy of the learning algorithm. More specifically, if we have prior knowledge of the variance of each term in the stochastic input vector to the network, we can take advantage of this

(1) PCA is also known as the Hotelling transform in digital image processing and the Karhunen-Loeve transformation in communication theory.

Part of the SPIE Conference on Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation • San Diego, California • July 1998
SPIE Vol. 3455 • 0277-786X/98/$10.00

information using the EGHA and improve the results, i.e., improved accuracy of the principal eigenvectors. This is essentially allowing "customized" learning rate parameters to be used for the extraction of each principal component. This capability is not available in the standard GHA. The approach that is taken here first extends the Karhunen-Oja symmetric subspace algorithm (KOSSA), i.e., the EKOSSA, and then from these results the EGHA is developed.

1.2 Overview of Principal Component Analysis (PCA)

PCA is a statistical method that determines an optimal linear transformation $y = Wx$ for a given input vector x, considered to be a zero-mean, wide-sense stationary, vector-stochastic process [15, 16, 19]. The matrix W is comprised of orthonormal eigenvectors associated with the input covariance matrix $C_x = \mathrm{E}[xx^T]$. Specifically, the *rows* of W are the principal eigenvectors associated with C_x. PCA transforms a large amount of correlated data into a set of statistically decorrelated components (principal components), i.e., PCA projects the input data from an n-dimensional space onto an m-dimensional output space, typically $m<<n$. The output space has a covariance matrix that is diagonal. The goal of the PCA is to find directions or the principal eigenvectors $w_1, w_2, \cdots, w_m$.

Let

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_m^T \end{bmatrix} \in \Re^{mxn}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \Re^{nx1}, \; y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \in \Re^{mx1} \tag{1}$$

The first m eigenvectors $\{w_1, w_2, \cdots, w_m\}$ of C_x are considered the m principal eigenvectors of C_x. These can be found from the standard eigenvalue problem

$$C_x w_j = \lambda_j w_j \quad \text{for } j = 1, 2, \cdots, n \tag{2}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. These eigenvalues are ordered from the largest to the smallest. The principal eigenvector w_1 corresponds to the largest eigenvalue λ_1, and w_2 corresponds to the second largest eigenvalue λ_2, and so on. If it is assumed that we are interested in the j^{th} principal component, and we let y_j be the weighted linear combination of the elements of the data input vector x, then we can write

$$y_j = w_{1j}x_1 + w_{2j}x_2 + \cdots + w_{nj}x_n \tag{3}$$

The scalar component (i.e., the j^{th} principal component) y_j is considered a stochastic variable with an associated variance given by

$$\mathrm{E}[y_j^2] = \sigma_{y_j}^2 = w_j^T C_x w_j \tag{4}$$

We require w_j to be of unit length, i.e.,

$$w_j^T w_j = \| w_j \|_2^2 = 1 \tag{5}$$

where $\| \bullet \|_2$ is the L_2-norm (or Euclidean norm). Having determined the eigenvalue λ_j, the eigenvector w_j can be found from

$$(C_x - \lambda_j I) w_j = 0 \tag{6}$$

Pre-multiplying both sides of (6) by w_j^T gives

$$w_j^T (C_x - \lambda_j I) w_j = w_j^T C_x w_j - \lambda_j w_j^T w_j = \sigma_{y_j^2} - \lambda_j = 0 \tag{7}$$

$$\therefore \sigma_{y_j}^2 = \lambda_j \tag{8}$$

For $j = 1$, $\sigma_{y_1}^2 = \lambda_1$ is the largest eigenvalue of C_x (the largest variance), and $w_1 = [w_{11} w_{12} \cdots w_{1n}]^T$ is the first principal eigenvector corresponding to the largest eigenvalue λ_1 of the input data. The vector w_1 indicates the direction in

the vector space associated with the largest variance. Therefore the linear transformation $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$ yields an output y which is the vector of principal components. Figure 1 shows PCA in a 2-dimensional vector space. The first principal component has the largest variance and retains the maximum information associated the inputs. In other words, the first principal component has the minimum projection error associate with it.

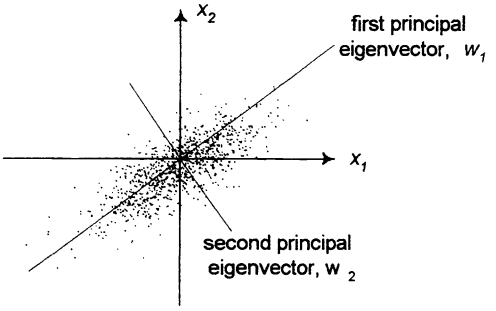

Figure 1: Geometrical interpretation of PCA.

2. PAST WORK

2.1 Oja's Learning Rule for Estimating a Single Principal Component (Normalized Hebbian Learning Algorithm)

Oja [18] proposed a single *linear* processing unit described by the equation $y_1 = \boldsymbol{w}_1^T \boldsymbol{x}$, see Fig. 2. The purpose of the learning algorithm is to find the first principal eigenvector $\boldsymbol{w}_1 = [w_{11}\, w_{12} \cdots w_{1n}]^T$. Starting with the representation-error vector defined as $\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}}$, we assume the length of the weighting vector is 1, i.e.,

$$\boldsymbol{w}_1^T \boldsymbol{w}_1 = \| \boldsymbol{w}_1 \|_2^2 = 1 \tag{9}$$

From the linear transformation $y_1 = \boldsymbol{w}_1^T \boldsymbol{x}$ and the estimate of x, i.e., $\hat{\boldsymbol{x}} = \boldsymbol{w}_1 y_1$, an energy function is formulated as follows

$$E(\boldsymbol{w}_1) = \frac{1}{2}\| \boldsymbol{e} \|_2^2 = \frac{1}{2}\| \boldsymbol{x} - \hat{\boldsymbol{x}} \|_2^2 = \frac{1}{2}(\boldsymbol{x} - \boldsymbol{w}_1 y_1)^T (\boldsymbol{x} - \boldsymbol{w}_1 y_1) \tag{10}$$

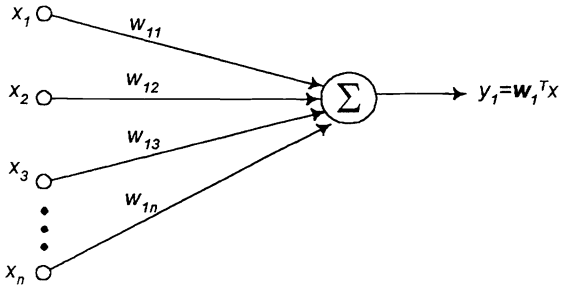

Figure 2: Oja's single neuron model.

Using this energy function, a continuous-time (analog) learning rule can be developed using the method of steepest descent as

$$\frac{d\boldsymbol{w}_1}{dt} = -\mu \nabla_{\boldsymbol{w}_1} E(\boldsymbol{w}_1) \tag{11}$$

Next we need to compute the gradient of (10) with respect to $\boldsymbol{w}_1$, i.e.,

$$\nabla_{w_1} E(\boldsymbol{w}_1) = \frac{\partial E(\boldsymbol{w}_1)}{\partial \boldsymbol{w}_1} = \frac{\partial}{\partial \boldsymbol{w}_1}\left[\frac{1}{2}(\boldsymbol{x} - \boldsymbol{w}_1 y_1)^T (\boldsymbol{x} - \boldsymbol{w}_1 y_1)\right] \tag{12}$$

By applying the following general results (for appropriately dimensioned A, B, and C)

$$\frac{\partial}{\partial \boldsymbol{A}} tr[\boldsymbol{BAC}] = \boldsymbol{B}^T \boldsymbol{C}^T$$
$$\frac{\partial}{\partial \boldsymbol{A}} tr[\boldsymbol{BA}^T \boldsymbol{C}] = \boldsymbol{CB} \tag{13}$$

and the appropriate chain rule, we obtain the gradient of $E(\boldsymbol{w}_1)$ as

$$\nabla_{w_1} E(\boldsymbol{w}_1) = \frac{1}{2}(-2\boldsymbol{x}y_1 + 2\boldsymbol{w}_1 y_1^2) = -\boldsymbol{x}y_1 + \boldsymbol{w}_1 y_1^2 \tag{14}$$

Therefore, Oja's continuous-time learning rule for a single neuron is given by

$$\frac{d\boldsymbol{w}_1}{dt} = \mu y_1 (\boldsymbol{x} - \boldsymbol{w}_1 y_1) \tag{15}$$

where $y_1 = \boldsymbol{w}_1^T \boldsymbol{x}$. The discrete-time version of (15) is given by

$$\boldsymbol{w}_1(k+1) = \boldsymbol{w}_1(k) - \mu \nabla_{w_1} E(\boldsymbol{w}_1) = \boldsymbol{w}_1(k) + \mu y_1(k)\{\boldsymbol{x}(k) - \boldsymbol{w}_1(k) y_1(k)\} \tag{16}$$

where $y_1(k) = \boldsymbol{w}_1^T(k)\boldsymbol{x}(k)$, and k is the discrete-time index. The scalar form of Oja's discrete-time learning rule in (16) is given by

$$w_{1j}(k+1) = w_{1j}(k) + \mu y_1(k)\{x_j(k) - w_{1j}(k) y_1(k)\} \tag{17}$$

where $j = 1, 2, \cdots, m$, and $\mu = \mu(k) > 0$ is the learning rate. Note that the $y_1(k)x_j(k)$ term in (17) is the typical Hebbian *co-occurrence* term [22].

Oja's learning rule converges to the weight vector w_1, which is the first principal eigenvector of $\boldsymbol{C}_x$ for the zero-mean input vector x, and y_1 is the first principal component. It is also true that w_1 maximizes the variance of the output y_1. Therefore, the single linear neuron trained by the Hebbian learning in (17) is a principal component analyzer of the input signal.

2.3 Karhunen-Oja Symmetric Subspace Learning Rule for Extraction of Multiple Principal Components

Karhunen and Oja [19, 20] proposed a single-layer neural network with m linear processing units described by the transformation $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$, see Fig. 3. The purpose of this architecture is to extract several principal components, $y_1, y_2, \cdots, y_m$, where $m << n$. The learning rule can be derived by first formulating an energy function given by

$$E(\boldsymbol{W}) = \frac{1}{2}\|\boldsymbol{e}\|_2^2 \tag{18}$$

where e is the representation error given by

$$\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{W}^T \boldsymbol{y} = \boldsymbol{x} - \boldsymbol{W}^T \boldsymbol{W}\boldsymbol{x} \tag{19}$$

Therefore, the energy function can be written as

$$E(\boldsymbol{W}) = \frac{1}{2}\|\boldsymbol{e}\|_2^2 = \frac{1}{2}\boldsymbol{e}^T\boldsymbol{e} = \frac{1}{2}(\boldsymbol{x}^T - \boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W})(\boldsymbol{x} - \boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}) \tag{20}$$

$$= \frac{1}{2}(\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}) \tag{21}$$

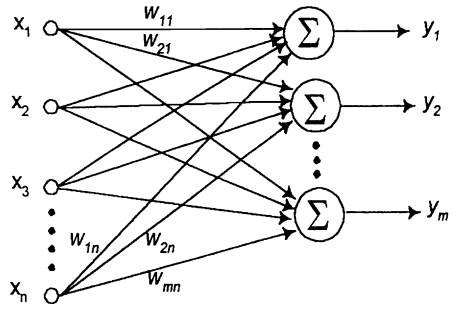

Figure 3: Neural architecture for multiple principal component extraction.

Using this energy function, a continuous-time (analog) learning rule can be developed as

$$\frac{\mathrm{d}\boldsymbol{W}}{\mathrm{d}t} = -\mu\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) \tag{22}$$

The gradient of (20) must be computed, i.e.,

$$\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) = \frac{\partial E(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{\partial}{\partial \boldsymbol{W}}\left[\frac{1}{2}(\boldsymbol{x}^T\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x})\right] \tag{23}$$

By applying the general results in (13), (14) and

$$\frac{\partial}{\partial \boldsymbol{A}} tr[\boldsymbol{B}\boldsymbol{A}^T\boldsymbol{C}\boldsymbol{A}] = \frac{\partial}{\partial \boldsymbol{A}} tr[\boldsymbol{B}\boldsymbol{A}^T\boldsymbol{C}\boldsymbol{A}\boldsymbol{I}] = \boldsymbol{C}\boldsymbol{A}\boldsymbol{B} + (\boldsymbol{B}\boldsymbol{A}^T\boldsymbol{C})^T = \boldsymbol{C}\boldsymbol{A}\boldsymbol{B} + \boldsymbol{C}^T\boldsymbol{A}\boldsymbol{B}^T \tag{24}$$

the gradient in (22) is

$$\begin{aligned}\nabla_{\boldsymbol{W}} E(\boldsymbol{W}) &= \frac{1}{2}\left[-2(2\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T) + (2\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W} + 2\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T)\right] \\ &= -\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T + \boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W} - \boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T + \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\end{aligned} \tag{25}$$

However, the last two terms will approach zero very quickly because $\boldsymbol{W}\boldsymbol{W}^T \to \boldsymbol{I} \in \Re^{mxm}$. Therefore, continuous-time learning rule is given by

$$\frac{d\boldsymbol{W}}{dt} = \mu(\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T - \boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}) \tag{26}$$

The discrete-time learning of (26) is given by

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + \mu(k)\{\boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k) - \boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k)\boldsymbol{W}^T(k)\boldsymbol{W}(k)\} \tag{27}$$

$$= \boldsymbol{W}(k) + \mu(k)\boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}(k)^T\{\boldsymbol{I} - \boldsymbol{W}^T(k)\boldsymbol{W}(k)\} \tag{28}$$

The learning rule given in (28) is known as the Karhunen-Oja symmetric subspace algorithm (KOSSA).

The scalar form of the KOSSA can be derived from (28) as follows. The second term on the right side of (28) can be written as (excluding the learning rate parameter and dropping the dependence on k).

$$\boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k) - \boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k)\boldsymbol{W}^T(k)\boldsymbol{W}(k)$$

$$= \begin{bmatrix} \boldsymbol{w}_1^T\boldsymbol{x} \\ \boldsymbol{w}_2^T\boldsymbol{x} \\ \vdots \\ \boldsymbol{w}_m^T\boldsymbol{x} \end{bmatrix}\left\{\boldsymbol{x}^T - [\boldsymbol{x}^T\boldsymbol{w}_1 \quad \boldsymbol{x}^T\boldsymbol{w}_2 \quad \cdots \quad \boldsymbol{x}^T\boldsymbol{w}_m]\begin{bmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_m^T \end{bmatrix}\right\} = y_i(k)\left[x_j(k) - \sum_{h=1}^{m} w_{hj}(k)y_h(k)\right] \tag{29}$$

for $i = 1, 2, \cdots, m$ and $j = 1, 2, \cdots, n$. Therefore, the scalar form of the KOSSA can be written as

$$w_{ij}(k+1) = w_{ij}(k) + \mu(k) y_i(k)[x_j(k) - \sum_{h=1}^{m} w_{hj}(k) y_h(k)] \tag{30}$$

where the learning rate is given by

$$\mu(k) = \frac{1}{\frac{\sigma}{\mu(k-1)} + \| y(k) \|_2^2} \tag{31}$$

where $\mu(0) = \frac{1}{\| y(0) \|_2^2}$, for $k = 1, 2, 3, \cdots$, and the forgetting factor is $0 \le \sigma \le 1$. If m = 1, the KOSSA reduces to Oja's single neuron learning rule. An interesting aspect of the KOSSA is the rows of the weight matrix $\boldsymbol{W}$ do not converge to the actual principal eigenvectors of C_x, but they do converge to some linear combination of the first m principal eigenvectors of C_x. Therefore, the neural network is able to learn the sub-space spanned by the first m principal eigenvectors.

2.1 Sanger's Learning Rule: Generalized Hebbian Algorithm (GHA)

Sanger's GHA allows the first m "true" principal eigenvectors to be determined. The GHA can be derived from the KOSSA [19]. Starting with (29) (dropping the dependence on k)

$$\boldsymbol{W x x}^T - \boldsymbol{W x x}^T \boldsymbol{W}^T \boldsymbol{W} = \boldsymbol{y x}^T - \boldsymbol{y y}^T \boldsymbol{W} = \boldsymbol{y x}^T - \boldsymbol{y y}^T \begin{bmatrix} \boldsymbol{w_1^T} \\ \boldsymbol{w_2^T} \\ \vdots \\ \boldsymbol{w_m^T} \end{bmatrix} \tag{32}$$

We want to "break" the symmetry of outer product $\boldsymbol{yy}^T$ in (33). This can be accomplished by only retaining the lower triangular portion of the symmetric matrix $\boldsymbol{yy}^T$, i.e., we want to apply the operator LT[$\boldsymbol{yy}^T$] (where LT[•] selects the lower triangular portion of the matrix). Apply this operator to $\boldsymbol{yy}^T$ in (33) we obtain

$$\boldsymbol{yx}^T - \mathrm{LT}\begin{bmatrix} y_1y_1 & y_1y_2 & \cdots & y_1y_m \\ y_2y_1 & y_2y_2 & \cdots & y_2y_m \\ \vdots & \vdots & \vdots & \vdots \\ y_my_1 & y_my_2 & \cdots & y_my_m \end{bmatrix}\begin{bmatrix} \boldsymbol{w_1^T} \\ \boldsymbol{w_2^T} \\ \vdots \\ \boldsymbol{w_m^T} \end{bmatrix} = \boldsymbol{yx}^T - \begin{bmatrix} y_1y_1 & 0 & \cdots & 0 \\ y_2y_1 & y_2y_2 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ y_my_1 & y_my_2 & \cdots & y_my_m \end{bmatrix}\begin{bmatrix} \boldsymbol{w_1^T} \\ \boldsymbol{w_2^T} \\ \vdots \\ \boldsymbol{w_m^T} \end{bmatrix}$$

$$= \boldsymbol{yx}^T - \begin{bmatrix} y_1y_1\boldsymbol{w}_1^T \\ y_2(y_1\boldsymbol{w}_1^T + y_2\boldsymbol{w}_2^T) \\ \vdots \\ y_m(y_1\boldsymbol{w}_1^T + y_2\boldsymbol{w}_2^T + \cdots + y_m\boldsymbol{w}_m^T) \end{bmatrix}\begin{bmatrix} \boldsymbol{w_1^T} \\ \boldsymbol{w_2^T} \\ \vdots \\ \boldsymbol{w_m^T} \end{bmatrix} = y_i(k)\left[x_j(k) - \sum_{h=1}^{i} w_{hj}(k) y_h(k) \right] \tag{33}$$

Therefore, the scalar form of the GHA can be written as

$$w_{ij}(k+1) = w_{ij}(k) + \mu(k) y_i(k)\left[x_j(k) - \sum_{h=1}^{i} w_{hj}(k) y_h(k) \right] \tag{34}$$

Comparing this scalar form of the GHA to the scalar form of KOSSA, we see the only difference is the upper limit of the summation. This algorithm is considered a detector of orthogonal features which encodes mutually independent aspects of the information contained in the large amount of input data.

3. NEW APPROACH

3.1 Derivation of the Extended Karhunen-Oja Symmetric Subspace Algorithm (EKOSSA) and the Extended GHA (EGHA)

In this section, the extended KOSSA (EKOSSA) is first derived, and from this result the extended GHA (EGHA) follows. We introduce a real positive definite, symmetric weighting matrix S, in the energy function given (18) as follows

$$E(\boldsymbol{W}) = \frac{1}{2}\boldsymbol{e}^T\boldsymbol{S}\boldsymbol{e} \tag{35}$$

where $\boldsymbol{e} \in \Re^{nx1}$ is the representation error, and $\boldsymbol{S} \in \Re^{nxn}$, $\boldsymbol{S} > 0$ and $\boldsymbol{S}^T = \boldsymbol{S}$. The representation error can be written as

$$\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x} \tag{36}$$

where $\hat{\boldsymbol{x}} = \boldsymbol{W}^T\boldsymbol{y}$, $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$. Therefore, the energy function can be written as

$$E(\boldsymbol{W}) = \frac{1}{2}\boldsymbol{e}^T\boldsymbol{S}\boldsymbol{e} = \frac{1}{2}(\boldsymbol{x}^T - \boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W})\,\boldsymbol{S}\,(\boldsymbol{x} - \boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}) \tag{37}$$

Using this energy function, a continuous-time (analog) learning rule can be developed using a steepest descent approach given by

$$\frac{\mathrm{d}\boldsymbol{W}}{\mathrm{d}t} = -\mu\nabla_{\boldsymbol{W}}E(\boldsymbol{W}) \tag{38}$$

where

$$\nabla_{\boldsymbol{W}}E(\boldsymbol{W}) = \frac{\partial}{\partial \boldsymbol{W}}E(\boldsymbol{W}) = \frac{\partial}{\partial \boldsymbol{W}}\left[\frac{1}{2}\boldsymbol{x}^T\boldsymbol{S}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{S}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{S}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{S}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}\right] \tag{39}$$

By applying the general results in (13) and the appropriate chain rule in (24), we can write the gradient of $E(W)$ as

$$\begin{aligned}\nabla_{\boldsymbol{W}}E(\boldsymbol{W}) &= \frac{1}{2}\left[-2\boldsymbol{W}\boldsymbol{S}\boldsymbol{x}\boldsymbol{x}^T - 2\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{S} + 2\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{S} + 2\boldsymbol{W}\boldsymbol{S}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\right] \\ &= -\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{S} + \boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{S} - \boldsymbol{W}\boldsymbol{S}\boldsymbol{x}\boldsymbol{x}^T + \boldsymbol{W}\boldsymbol{S}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T \\ &= [-\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T + \boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}]\boldsymbol{S} - \boldsymbol{W}\boldsymbol{S}\boldsymbol{x}\boldsymbol{x}^T + \boldsymbol{W}\boldsymbol{S}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\end{aligned} \tag{40}$$

If $\boldsymbol{S} = \boldsymbol{I}$ in (40), this leads to the same results shown in (28). Therefore, the EKOSSA is a more general result than the KOSSA. However, if $\boldsymbol{S} \neq \boldsymbol{I}$, the terms that were discarded in (25) must be retained in (40), and the discrete-time learning rule is given by

$$\begin{aligned}\boldsymbol{W}(k+1) &= \boldsymbol{W}(k) - \mu(k)\nabla E(\boldsymbol{W}) \\ &= \boldsymbol{W}(k) + \mu(k)[\overbrace{\boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k)\boldsymbol{S}(k) - \boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k)\boldsymbol{W}^T(k)\boldsymbol{W}(k)\boldsymbol{S}(k)}^{part\ I} \\ &\quad + \underbrace{\boldsymbol{W}(k)\boldsymbol{S}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k) - \boldsymbol{W}(k)\boldsymbol{S}(k)\boldsymbol{W}^T(k)\boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k)}_{part\ II}] \\ &= \boldsymbol{W}(k) + \mu(k)\{\boldsymbol{W}(k)\boldsymbol{x}(k)\boldsymbol{x}^T(k)[\boldsymbol{I} - \boldsymbol{W}^T(k)\boldsymbol{W}(k)]\boldsymbol{S}(k) + \boldsymbol{W}(k)\boldsymbol{S}(k)[\boldsymbol{I} - \boldsymbol{W}^T(k)\boldsymbol{W}(k)]\boldsymbol{x}(k)\boldsymbol{x}^T(k)\}\end{aligned} \tag{41}$$

Equation (41) is the vector-matrix form of the extended KOSSA (EKOSSA). Now by defining

$$\boldsymbol{Q}(k) = \boldsymbol{x}(k)\boldsymbol{x}^T(k)[\boldsymbol{I} - \boldsymbol{W}^T(k)\boldsymbol{W}(k)]\boldsymbol{S}(k) \tag{42}$$

$$\boldsymbol{Q}^T(k) = \boldsymbol{S}(k)[\boldsymbol{I} - \boldsymbol{W}^T(k)\boldsymbol{W}(k)]\boldsymbol{x}(k)\boldsymbol{x}^T(k) \tag{43}$$

a simplified form of the EKOSSA can be written as

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) - \mu(k)\boldsymbol{W}(k)[\boldsymbol{Q}(k) + \boldsymbol{Q}^T(k)] \tag{44}$$

The EGHA can now be derived from (41). *Part I* in (41) can be written as (dropping the dependence on k)

$$\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{S} - \boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{S} = [\boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T - \boldsymbol{W}\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{W}^T\boldsymbol{W}]\boldsymbol{S} = \{\boldsymbol{y}\boldsymbol{x}^T - \boldsymbol{y}\boldsymbol{y}^T \begin{bmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_m^T \end{bmatrix}\} \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nn} \end{bmatrix} \tag{45}$$

If we break the symmetry on the outer product matrix $\boldsymbol{y}\boldsymbol{y}^T$ in (45) using the lower triangular operator LT[•], we obtain

$$\{\boldsymbol{y}\boldsymbol{x}^T - \mathrm{LT}\begin{bmatrix} y_1y_1 & y_1y_2 & \cdots & y_1y_m \\ y_2y_1 & y_2y_2 & \cdots & y_2y_m \\ \vdots & \vdots & \vdots & \vdots \\ y_my_1 & y_my_2 & \cdots & y_my_m \end{bmatrix}\begin{bmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_m^T \end{bmatrix}\} \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nn} \end{bmatrix}$$

$$= \{\boldsymbol{y}\boldsymbol{x}^T - \begin{bmatrix} y_1y_1\boldsymbol{w}_1^T \\ y_2(y_1\boldsymbol{w}_1^T + y_2\boldsymbol{w}_2^T) \\ \vdots \\ y_m(y_1\boldsymbol{w}_1^T + y_2\boldsymbol{w}_2^T + \cdots + y_m\boldsymbol{w}_m^T) \end{bmatrix}\begin{bmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_m^T \end{bmatrix}\} \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nn} \end{bmatrix}$$

$$= y_i(k)\sum_{g=1}^{n}\left[x_g(k) - \sum_{h=1}^{i} w_{hg}(k)y_h(k) \right]s_{gj}(k) \tag{46}$$

In *part II* of (41), if we break the symmetry on the outer product matrix $\boldsymbol{x}\boldsymbol{x}^T$ using the upper triangular operator UT[•], we obtain

$$\underbrace{\boldsymbol{W}\boldsymbol{S}(\mathrm{UT}[\boldsymbol{x}\boldsymbol{x}^T])}_{part\,1} - \underbrace{\boldsymbol{W}\boldsymbol{S}\boldsymbol{W}^T\boldsymbol{W}(\mathrm{UT}[\boldsymbol{x}\boldsymbol{x}^T])}_{part\,2} \tag{47}$$

Part 1 in (47) can be written as

$$\boldsymbol{W}\boldsymbol{S}(\mathrm{UT}[\boldsymbol{x}\boldsymbol{x}^T])$$

$$= \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}\begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ s_{n1} & s_{n2} & \cdots & s_{nn} \end{bmatrix}\begin{bmatrix} x_1x_1 & x_1x_2 & \cdots & x_1x_n \\ 0 & x_2x_2 & \cdots & x_2x_n \\ 0 & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & x_nx_n \end{bmatrix}$$

$$= \begin{bmatrix} \sum_{g=1}^{n} w_{1g}s_{g1}x_1x_1 & \sum_{f=1}^{2}\sum_{g=1}^{n} w_{1g}s_{gf}x_fx_2 & \cdots & \sum_{f=1}^{n}\sum_{g=1}^{n} w_{1g}s_{gf}x_fx_n \\ \sum_{g=1}^{n} w_{1g}s_{g1}x_1x_1 & \sum_{f=1}^{2}\sum_{g=1}^{n} w_{2g}s_{gf}x_fx_2 & \cdots & \sum_{f=1}^{n}\sum_{g=1}^{n} w_{2g}s_{gf}x_fx_n \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{g=1}^{n} w_{mg}s_{g1}x_1x_1 & \sum_{f=1}^{2}\sum_{g=1}^{n} w_{mg}s_{gf}x_fx_2 & \cdots & \sum_{f=1}^{n}\sum_{g=1}^{n} w_{mg}s_{gf}x_fx_n \end{bmatrix}$$

$$= \sum_{f=1}^{j}\sum_{g=1}^{n} w_{ig}s_{gf}x_fx_j \tag{48}$$

and *part 2* can be written as

$$\boldsymbol{W}\boldsymbol{S}\boldsymbol{W}^T\boldsymbol{W}(\mathrm{UT}[\boldsymbol{x}\boldsymbol{x}^T]) =$$

$$\left[\sum_{g=1}^{n} w_{ig}s_{gj}\right]\begin{bmatrix} \sum_{h=1}^{m} w_{h1}w_{h1} & \sum_{h=1}^{m} w_{h1}w_{h2} & \cdots & \sum_{h=1}^{m} w_{h1}w_{hn} \\ \sum_{h=1}^{m} w_{h2}w_{h1} & \sum_{h=1}^{m} w_{h2}w_{h2} & \cdots & \sum_{h=1}^{m} w_{h2}w_{hn} \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{h=1}^{m} w_{hn}w_{h1} & \sum_{h=1}^{m} w_{hn}w_{h2} & \cdots & \sum_{h=1}^{m} w_{hn}w_{hn} \end{bmatrix}\begin{bmatrix} x_1x_1 & x_1x_2 & \cdots & x_1x_n \\ 0 & x_2x_2 & \cdots & x_2x_n \\ 0 & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & x_nx_n \end{bmatrix}$$

$$= \left[\sum_{g=1}^{n} w_{ig}s_{gj}\right]\left[\sum_{f=1}^{j}\sum_{h=1}^{m} w_{hj}w_{hf}x_f x_j\right] \tag{49}$$

Therefore, combining (48) and (49), *part II* in (41) can be written as

$$\boldsymbol{WS}(\mathrm{UT}[\boldsymbol{x}\boldsymbol{x}^T]) - \boldsymbol{WSW}^T\boldsymbol{W}(\mathrm{UT}[\boldsymbol{x}\boldsymbol{x}^T])$$

$$= \sum_{f=1}^{j}\sum_{g=1}^{n} w_{ig}s_{gf}x_f x_j - \left[\sum_{g=1}^{n} w_{ig}s_{gj}\right]\left[\sum_{f=1}^{j}\sum_{h=1}^{m} w_{hj}w_{hf}x_f x_j\right] \tag{50}$$

Therefore, from (46) and (50), the scalar form of the EGHA can be written as

$$w_{ij}(k+1) = w_{ij}(k) + \mu(k)\{y_i(k)\sum_{g=1}^{n}\left[x_g(k) - \sum_{h=1}^{i} w_{hg}(k)y_h(k)\right]s_{gj}(k)$$

$$+ \sum_{f=1}^{j}\sum_{g=1}^{n} w_{ig}(k)s_{gf}(k)x_f(k)x_j(k) - \left[\sum_{g=1}^{n} w_{ig}(k)s_{gj}(k)\right]\left[\sum_{f=1}^{j}\sum_{h=1}^{m} w_{hj}(k)w_{hf}(k)x_f(k)x_j(k)\right]\} \tag{51}$$

If the weighting matrix S is diagonal, (51) can be simplified as

$$w_{ij}(k+1) = w_{ij}(k) + \mu(k)\{y_i(k)\left[x_j(k) - \sum_{h=1}^{i} w_{hj}(k)y_h(k)\right]s_{jj}(k)$$

$$+ \sum_{f=1}^{j} w_{if}(k)s_{ff}(k)x_f(k)x_j(k) - \left[\sum_{g=1}^{n} w_{ig}(k)s_{gj}(k)\right]\left[\sum_{f=1}^{j}\sum_{h=1}^{m} w_{hj}(k)w_{hf}(k)x_f(k)x_j(k)\right]\} \tag{52}$$

where the learning rate parameter is given by

$$\mu(k) = \frac{1}{\dfrac{\sigma}{\mu(k-1)} + \|\boldsymbol{y}(k)\|_2^2} \tag{53}$$

and the forgetting factor must lie in the range $0 \le \sigma \le 1$.

3.2 Simulation Results

Comparison of the EGHA with Sanger's GHA

In this section the performance of the EGHA is compared to Sanger's GHA. The speed of convergence and the overall error performances are compared. In each of the three cases studied, the same input data, initial weighting matrix $W(0)$, and forgetting factor ($\sigma = 0.9$) in the adjustable learning rate parameter were used. In each test it was assumed that we had prior knowledge of the variance of each term in the stochastic input vector to the network. Four different scenarios were considered in this comparative analysis: (i) Matlab eigenanalysis, using the "eig" built-in function, (ii) GHA, (iii) EGHA ($S=I$), and (iv) EGHA with a diagonal S matrix. The diagonal elements of the weighting matrix S for scenario (iv) are the reciprocals of the standard deviations of the input data, i.e., $s_{ii} = 1/\sigma_i$, for $i = 1, 2, \cdots, n$.

In the first test case 5000 zero-mean Gaussian 3-dimensional vectors were generated. The respective variances of the components in the stochastic vector are set at $\sigma_1^2 = 100$, $\sigma_2^2 = 25$ and $\sigma_3^2 = 1$. The objective is to estimate all three principal eigenvectors and compare the results to the results computed using the Matlab built-in function "eig." The results obtained using the Matlab function are considered the "truth" values (i.e., the true principal eigenvectors and eigenvalues), and the results obtained using the other methods are compared to them. To compute the Matlab results, the covariance matrix is first estimated from the 5000 random vectors, i.e.,

$$C_x \cong \frac{1}{\mathrm{N}} \sum_{k=1}^{\mathrm{N}} x(k)x^T(k) = \frac{1}{\mathrm{N}} XX^T \tag{54}$$

where N=5000 and

$$X = [x(1), x(2), \cdots, x(\mathrm{N})] \tag{55}$$

In Table 1, the first column shows the Matlab eigenanalysis results, i.e., computing the eigenvalues and eigenvectors of C_x from (54). The three neural network approaches discussed above, i.e., scenarios (ii), (iii), and (iv), were used next to determine the principal eigenvectors (and the associated eigenvalues). These results are shown in the next three columns of Table 1. The eigenvalues for these three cases are estimated according to

$$\lambda_i \cong var(w_i^T X) \tag{56}$$

where w_i is the i^{th} principal eigenvector adaptively determined using one of the methods and X is given in (55). Table 1 shows that the EGHA with the weighting matrix given by S=diag($1/\sigma_i$), for $i = 1, 2, \cdots, n$, yields the best results even though more training epochs were required for convergence. The performance for each scenario is based on the sum of the absolute values of the errors for each estimated eigenvalue compared to the Matlab results, i.e.,

$$\text{sum of errors} = \sum_{i=1}^{n} \left| \lambda_i^M - \lambda_i^{NN} \right| \tag{57}$$

where λ_i^M is the i^{th} eigenvalue computed by using the Matlab built-in function "eig" and λ_i^{NN} is the i^{th} eigenvalue estimated by one of the neural network methods.

Table 1. Performance Results for First Experiment. ($\sigma_1^2 = 100, \sigma_2^2 = 25, \sigma_3^2 = 1$)

(i) Eigenanalysis: eig(C_x)	(ii) GHA	(iii) EGHA (S=I)	(iv) EGHA (S=diag($1/\sigma_i$))
V(eigenvectors)= 1.0000 -0.0026 -0.0002 0.0026 1.0000 0.0009 -0.0002 0.0009 -1.0000 D(eigenvalues) = 102.0120 0 0 0 25.5471 0 0 0 1.0177	2 epochs to converge W^T = 0.9920 0.0564 -0.0134 0.1557 -1.0074 -0.0216 -0.0008 -0.0204 0.9972 eigenvalues = 101.0004 26.2369 1.0421 sum of errors = 1.7312	2 epochs to converge W^T = 0.9807 0.1200 -0.0315 0.3181 -0.9898 -0.0384 0.0005 -0.0420 0.9872 eigenvalues = 100.7516 26.4626 1.1305 sum of errors = 2.2881	14 epochs to converge W^T = 1.0005 -0.0216 -0.0007 0.0476 -1.0000 0.0122 -0.0009 -0.0256 -1.0004 eigenvalue = 102.1162 25.6125 1.0219 sum of errors = 0.1792

Table 2 shows the test results for the second experiment. The data are generated in the same way as in the first experiment except the variances used now are: $\sigma_1^2 = 10$, $\sigma_2^2 = 2$ and $\sigma_3^2 = 1$. All three neural network approaches required one training epoch to converge. The EGHA with S=diag($1/\sigma_i$) again had the best performance compared to the other two approaches.

Table 2. Performance Results for Second Experiment. ($\sigma_1^2 = 10, \sigma_2^2 = 2, \sigma_3^2 = 1$)

(i) Eigenanalysis: eig(C_x)	(ii) GHA	(iii) EGHA (S=I)	(iv) EGHA (S=diag($1/\sigma_i$))
	1 epoch to converge	1 epoch to converge	1 epoch to converge
V(eigenvectors) =	W^T =	W^T =	W^T =
1.0000 -0.0022 -0.0006	0.9949 0.0472 -0.0290	0.9862 0.0916 -0.0442	0.9979 0.0143 -0.0168
0.0022 1.0000 0.0059	0.1401 -1.0011 -0.1080	0.2815 -0.9694 -0.2241	0.1628 -0.9797 -0.1983
-0.0005 0.0059 -1.0000	-0.0119 -0.1209 0.9952	-0.0112 -0.2689 0.9575	-0.0127 -0.2400 0.9758
D(eigenvalues) =	eigenvalues =	eigenvalues =	eigenvalues =
10.2012 0 0	10.1360	10.0864	10.2119
0 2.0438 0	2.0861	2.0804	2.0252
0 0 1.0176	1.0396	1.0540	1.0504
	sum of errors = 0.1295	sum of errors = 0.1878	sum of errors = 0.0621

Table 3 shows the test results for the third experiment. Again the data are generated the same as before except the variances used now are: $\sigma_1^2 = 100$, $\sigma_2^2 = 50$ and $\sigma_3^2 = 1$. Case (ii) required 3 training epoch and case (iii) required 4 training epochs to converge. Case (iv) required more epochs to converge than the other two cases, specifically 17 epochs, however, it produced the most accurate results. When a smaller forgetting factor was used ($\sigma = 0.7$) for the three different neural networks, case (iv) required 4 training epochs to converge and was again the most accurate. However, the accuracy was not as good as that shown in Table 3.

Table 3. Performance Results for Third Experiment. ($\sigma_1^2 = 100, \sigma_2^2 = 50, \sigma_3^2 = 1$)

(i) Eigenanalysis: eig(C_x)	(ii) GHA	(iii) EGHA (S=I)	(iv) EGHA (S=diag($1/\sigma_i$))
	3 epochs to converge	4 epochs to converge	17 epochs to converge
V(eigenvectors)=	W^T =	W^T =	W^T =
1.0000 -0.0055 -0.0002	0.9682 0.1331 -0.0149	0.9286 0.2848 -0.0397	1.0012 -0.0412 0.0049
0.0055 1.0000 0.0006	0.2626 -0.9969 -0.0126	0.5056 -0.9276 -0.0462	0.0156 -0.9990 -0.0059
-0.0002 0.0006 -1.0000	-0.0037 -0.0158 0.9869	-0.0038 -0.0324 0.9553	-0.0056 -0.0181 0.9987
D(eigenvalues) =	eigenvalues =	eigenvalues =	eigenvalues =
102.0131 0 0	99.2290	101.2304	102.2063
0 51.0937 0	52.5267	52.1025	51.1963
0 0 1.0177	1.0220	1.1981	1.0189
	sum of errors = 4.2214	sum of errors = 1.9719	sum of errors = 0.2970

4. CONCLUSIONS

In summary, for the three simulations that were run, both Sanger's GHA and the EGHA with S=I had the fastest convergence, however, the EGHA with S=diag($1/\sigma_i$) had the best accuracy. It appears from the results in Table 1 and 3 that the input data with the larger variances will require more training epochs for convergence. It was demonstrated in one example that a smaller forgetting factor improved the speed of convergence, but the accuracy of the results declined. One of the main advantages of the EGHA is that selected error terms can be weighted differently. Depending on prior knowledge of the input data, the weighting can involve statistical information relating to the input to the network. This was the case in scenario (iv) where the weighting matrix was diagonal. The diagonal elements were the reciprocals of the standard deviations of the input data. This EGHA performed the best in the three simulations that were run. Further work will involve incorporating robustness into the EGHA.

5. REFERENCES

1. Hotelling, H., "Analysis of a Complex of Statistical Variables into Principal Components," *J. of Educ. Psychol.*, **24**, pp. 417-441, 498-520, 1933.
2. Taylor, J.G & S. Coombes, "Learning Higher Order Correlations," *Neural Networks*, **6**(3), pp. 423-427, 1993.

3. Skarbek, W., A. Cichocki, & W. Kasprzak, "Principal Subspace Analysis for Incomplete Image Data in One Learning Epoch," *Neural Network World*, **6**(3), pp. 375-382, 1996.
4. Xu, L & A. Krzyzak, " Neural Nets for Dual Subspace Pattern Recognition Method," *International Journal of Neural Systems*, **2**(3), pp. 169-184, 1991.
5. Sirat, J.A., "A Fast Neural Algorithm for Principal Component Analysis and Singular Value Decomposition," *International Journal of Neural Systems*, **2**, pp. 147-155, 1991.
6. Diamantaras, K.I. & S.Y. Kung, *Principal Component Neural Networks*, John Wiley and Sons, 1996.
7. Oja, E., H. Ogawa, & J. Wangviwattana, "Principal Component Analysis by Homogeneous Neural Networks, Part I: The Weighted Subspace Criterion," *IEICE Trans. Inf. & Syst.*, **E75-D** (3), pp. 366-375, 1992.
8. Oja, E., H. Ogawa, & J. Wangviwattana, "Principal Component Analysis by Homogeneous Neural Networks, Part II: Analysis and Extensions of the Learning Algorithm," *IEICE Trans. Inf. & Syst.*, **E75-D** (3), pp. 376-382, 1992.
9. Karhunen, J. & J. Joutsensalo, "Representation and Separation of Signals Using Nonlinear PCA Type Learning," *Neural Networks*, **7**(1), pp. 113-127, 1994.
10. Karhunen, J. & J. Joutsensalo, "Generalization of Principal Component Analysis Optimization Problems, and Neural Networks," *Neural Networks*, **8**(4), pp. 549-562, 1995.
11. Solo, V. & X. Kong, "Performance Analysis of Adaptive Eigenanalysis Algorithms," *IEEE Transactions on Signal Processing*, **46**(3), pp. 636-646, 1998.
12. Karhunen, J. & J. Joutsensalo, "Learning of Robust Principal Component Subspace," *Proceedings of 1993 International Joint Conference on Neural Networks*, **3**, pp. 2409-2412, 1993.
13. Wang, C., H.C. Wu, & J.C. Principe, "A Cost Function for Robust Estimation of PCA," *Proceedings of SPIE*, **2760**, pp. 120-127, 1996.
14. Chen, L.H. & S. Chang, "An Adaptive Learning Algorithm for Principal Component Analysis," *IEEE Transactions on Neural Networks*, **6**(5), pp. 1255-1263, 1995.
15. Cichocki, A. & R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, John Wiley and Sons, 1993.
16. Strang, G., *Linear Algebra and Its Application*, Harcourt Brace Jovanovich, Inc, 1988.
17. Sanger, T.D., "Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network," *Neural networks*, **2**, pp. 459-473, 1989.
18. Oja, E., "A simplified Neuron Model as a Principal Component Analyzer," *Journal of Mathematical Biology*, **15**, pp. 267-273, 1982.
19. Oja, E. & J. Karhunen, "On Stochastic Approximation of the Eigenvectors and Eigenvalues of the Expectation of a Random Matrix," *Journal of Mathematical Analysis and Applications*, **106**, pp. 69-84, 1985.
20. Oja, E., "Principal Components, Minor Components, and Linear Neural Networks," *Neural Networks*, 5(6), pp. 927-935, 1992.
21. Plumbley, M.D., "Lyapunov Functions for Convergence of Principal Component Algorithms," *Neural Networks*, **8**(1), pp. 11-23, 1995.
22. Hertz, J., A. Krogh, & R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, 1991.
23. Softky,W.R. & D.M. Kammen, "Correlations in High Dimensional or Asymmetric Data Sets: Hebbian Neuronal Processing," *Neural Networks*, **4**(3), pp. 337-347, 1991.
24. Bouzerdoum, A. & T.R. Pattison, "Neural Network for Quadratic Optimization with Bound Constraints," *IEEE Transactions on Neural Networks*, **4**(2), pp. 293-304, 1993.
25. Baldi, P. & K. Hornik, "Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima," *Neural Networks*, **2**, pp. 53-58, 1989.
26. Rubner, J. & P. Tavan, "A Self-Organizing Network for Principal Component Analysis," *Europhysics Letters*, **10**(7), pp. 693-698, 1989.
27. Karayiannis, N.B., "Accelerating the Training of Feedforward Neural Networks Using Generalized Hebbian Rules for Initializing the Internal Representations," *IEEE Transactions on Neural Networks*, 7(2), pp. 419-426, 1996.
28. Diamantaras, K.I. & S.Y. Kung, "Cross-Correlation Neural Network Models," *IEEE Transactions on Signal Processing*, **42**(11), pp. 3218-3223, 1994.
29. Mathew, G. & V.U. Reddy, "Orthogonal Eigensubspace Estimation Using Neural Networks," *IEEE Transactions on Signal Processing*, **42**(7), pp. 1803-1811, 1994.
30. Yan, W.Y., U. Helmke, & J.B. Moore, "Global Analysis of Oja's Flow for Neural Networks," *IEEE Transactions on Neural Networks*, **5**(5), pp. 674-683, 1994.
31. Kambhatla, N. & T.K. Leen, "Dimension Reduction by Local Principal Component Analysis," *Neural Computation*, **9**, pp. 1493-1516, 1997.
32. Dehaene, J., M. Moonen, & J. Vandewalle, "An Improved Stochastic Gradient Algorithm for Principal Component Analysis and Subspace Tracking," *IEEE Transactions on Signal Processing*, 45(10), pp. 2582-2586, 1997.