Presented in *III Congresso Brasileiro de Redes Neurais*, Florianpolis, SC, Julho, 1997, pp. 7–11.

A New Training Algorithm to Reduce the Computational Complexity of Principal Component Analysis by Hebbian Learning[*]

Maria Cristina Felippetto De Castro
cristina@ee.pucrs.br

Fernando César C. De Castro
decastro@ee.pucrs.br

José Nelson Amaral
amaral@ee.pucrs.br

Paulo Roberto G. Franco
pfranco@ee.pucrs.br

Electrical Engineering Department
Pontifícia Universidade Católica do Rio Grande do Sul
90619-900 - Porto Alegre - RS - Brazil

Abstract

In this work we propose a technique that reduces the computational complexity of training an Artificial Neural Network (ANN) through Hebbian Learning. We restrict the learning process to the neurons that effectively are in the convergence process. Our results indicate that the proposed method can reduce the temporal complexity of the ANN training without degrading the Principal Components obtained(PCs).

1 Introduction

In this work we focus in the Generalized Hebbian Algorithm (GHA) proposed by Sanger in 1989, which combines the Gram-Schimidt orthonormalization [1] to the single linear neuron model introduced by Oja in 1982 [7]. ANNs with one single layer of linear neurons trained according to the Hebbian learning rule perform Principal Components Analysis (PCA) in an input data set [7, 8, 4] . After the convergence of the GHA algorithm, each neuron of the ANN yields:

(a) A set of synapse weights that are the eigenvector components associated to the respective eigenvalue of the input data set covariance matrix,

(b) An output value corresponding to the PC which represents the vectorial projection of the input data set over the associated eigenvector.

The PCA theory states that the set of the covariance matrix eigenvectors constitutes an orthonormal basis of the input data set [1]. The variance of the input data set projections are maximum at the eigenvectors directions, and the variance value is equal to the eigenvalue associated to that direction [7, 6].

During the convergence process the GHA extracts the PCs one by one, in variance descending order. In an ANN with m neurons N_j, $j = 0, 1, \ldots, m-1$, the synapses of the neuron N_0 converges to the eigenvector associated to the highest eigenvalue while the synapses of the neuron N_{m-1} converges to the eigenvector associated to the lowest eigenvalue.

It is well known that when GHA is applied, a neuron N_j will not converge until all neurons $N_k < N_j$ have converged. However the adaptation of N_j will start before the neurons that immeaditely precede it have converged. Also after

*This work was in part supported by a grant from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and by Pontifícia Universidade Católica do

converged, it would be a waste of computational resources to keep updating its weights. Moreover, because only few neurons, which are immediately subsequent to the converging neuron N_c, will be near the convergence point, the updating of the synapse weights of the whole set of neurons N_s, $c < s < m - 1$, also leads to unnecessary computations.

To avoid updating the synapses of all neurons that are not ready for convergence yet, we propose a new Training Window Algorithm (TWA). The TWA goal is to reduce the computational cost of the GHA training. This goal is achieved by applying the GHA only to a training window of size $W_s < m$ neurons. For instance, if the neuron N_c is the first neuron in the training window, the GHA is applied only to neurons $N_c, N_{c+1}, \ldots, N_{c+W_s-1}$. When the neuron N_c has converged, the window is slided down by incrementing the value N_c. This process is repeated until all m neurons have converged. Experimental results indicate that the TWA introduces a significant reduction of the GHA temporal complexity.

2 The Training Window Algorithm

In the experiments presented in this paper, we apply the TWA-GHA to digital image compression via PCA. We use images of $N \times N$ pixels, $N = 128$, quantized in 256 gray levels. The images are partitioned in $n_s = (N/8)^2$ frames of size $l \times l$ pixels, $l = 8$. This n_s frames, after properly transformed, will form the ANN training set. The pixels values of the n^{th} frame are normalized to the interval $[0, 1.0]$ and form frames M_n, $n = 0, 1, \ldots, n_s - 1$ [5].

The ANN is made of $p = 64$ nodes in the input layer and a single output layer with $m = 16$ linear neurons. The goal is to extract the 16 PCs of the original image.

The n^{th} vector $x(n)$ of the ANN training set is extracted from a frame M_n according to equation (1).

$$x_i(n) = M_n(a, b) - \overline{x_i} \tag{1}$$

where $x_i(n)$ refers to the i^{th} element of the vector $x(n)$ and

$$\overline{x_i} = \frac{1}{n_s} \sum_{n=0}^{n_s - 1} M_n(a, b) \tag{2}$$

$$i = a + lb \tag{3}$$

To train the ANN, first we initialize the synapse weights randomly with uniform probability density. The learning rate η is initially set to a value around 1×10^{-3}. The presentation of one complete training set to the ANN constitutes one epoch. The ANN training is performed by presenting the training set for several epochs until the neuron synapses of the first neuron in the training window converge to its associated eigenvector. For each vector of the training set presented to the ANN the synapse weights of the neurons inside the training window are updated according to equation (4) and the neuron output values are given by equation (5). Remember that a neuron j is inside the training window if and only if $N_c \leq j < N_c + W_s$.

$$\Delta w_{ji}(n) = \eta y_j(n)\{x_i(n) - \sum_{k=0}^{j} w_{ki}(n) y_k(n)\} \tag{4}$$

$$y_j(n) = \sum_{i=0}^{p-1} w_{ji}(n) x_i(n) \tag{5}$$

In order to shuffle the training set, at the end of each epoch, we randomly select a pair of vectors in the set and permute their positions. This operation is performed n_s times using a random number generator with uniform probability density.

In order to adequate the training process to the converging neuron requirements, the learning rate applied to the neurons of the training window is changed at the end of each epoch according to equation (6).

$$\eta = \frac{1/\lambda}{\alpha} \tag{6}$$

where λ is the eigenvalue associated to neuron N_c, and α is an arbitrary constant that usually lies in the interval [500, 2000] [2].

One iteration consists of presenting one vector of the training set to the ANN. Lets define $\overrightarrow{W}$ as the vector that has its components defined by the weight of the neuron synapses. In this work the first neuron of the training window is considered to have converged at iteration n if the norm of $\overrightarrow{W}$ remains approximately constant and unitary over the last 3 iterations. This convergence condition is achieved if the condition expressed in (7) holds true.

$$\frac{1}{3}\left\{\sum^{2} \frac{\| \overrightarrow{W}(n - \kappa - 1) \|}{\| \overrightarrow{W}(n - \kappa) \|}\right\} - 1 < 1 \times 10^{-4} \tag{7}$$

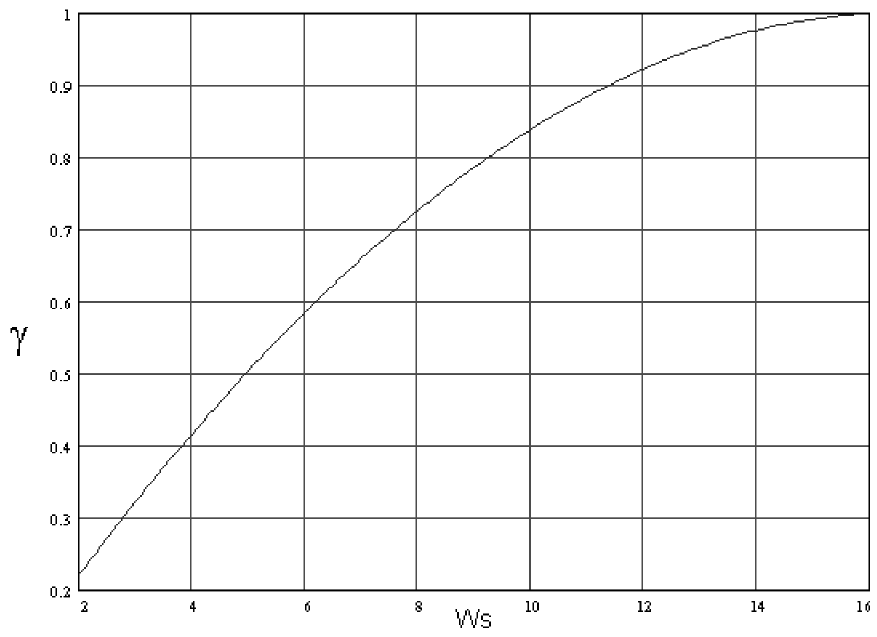

Figure 1: TWA comparative complexity performance for $m = 16$ and $p = 64$. γ is the TWA-GHA to single GHA complexity ratio and W_s is the training window size.

3 Reconstructing the Estimated Image

After the convergence of all the neurons in the output layer, the set of synaptic weight of vectors $\overrightarrow{W}_i$ is stored along with the output vectors $\overrightarrow{Y}_i$ generated by applying each one of the n_s vectors $\widehat{x}_i$ of the original image to the input of the ANN.

To reconstruct the image from the stored information, we must build the aproximations for the original frames $\widehat{M_n}(a,b)$. This reconstruction is yielded by equations (8) and (9),

$$\widehat{x}_i(n) = \sum_{j=0}^{r-1} y_j(n) q_{ji} + \overline{x_i} \qquad (8)$$

$$\widehat{M_n}(a,b) = 255 \times \widehat{x}_i(n) \qquad (9)$$

where

$$i = a + lb \qquad (10)$$

$$a = 0,1,\ldots,l-1 \quad b = 0,1,\ldots,l-1 \quad n = 0,1,\ldots,n_s-1 \qquad (11)$$

and 255 is the pixel denormalization factor.

In the equations refered above, $\widehat{x}_i(n)$ represents the reconstruction of the n^{th} input vector is the output of the j^{th} neuron to the n^{th} input vector and q_{ji} is the eigenvector associated to synapse vector $\overrightarrow{W}$ of the j^{th} converged neuron. The index $i = 0,1,\ldots,p-1$ represents the i^{th} element of the vectors $\widehat{x}(n)$, $\overline{x}$, and q_j.

The estimated original image is then reassembled from the estimated frame set. The $l \times l$ frames $\widehat{M_n}(a,b)$, $l = 8$, $n = 0,1,\ldots,n_s-1$, is assigned to the n^{th} respective frame in the 128×128 estimated original image.

4 The TWA Computational Complexity

In this section we study the GHA-TWA computational complexity compared to the complexity of the original GHA [7]. Given two ANNs with identical training sets, both ANNs with m neurons and p synapses per neuron. Lets train one of the ANNs with GHA and the other with TWA-GHA with a training window of size W_s.

In order to simplify the mathematics involved, we will compare the two methods based on two assumptions:

a. The difference between the number of epochs necessary to the convergence of neurons N_c and N_c+1 is a constant number ϵ, where N_c

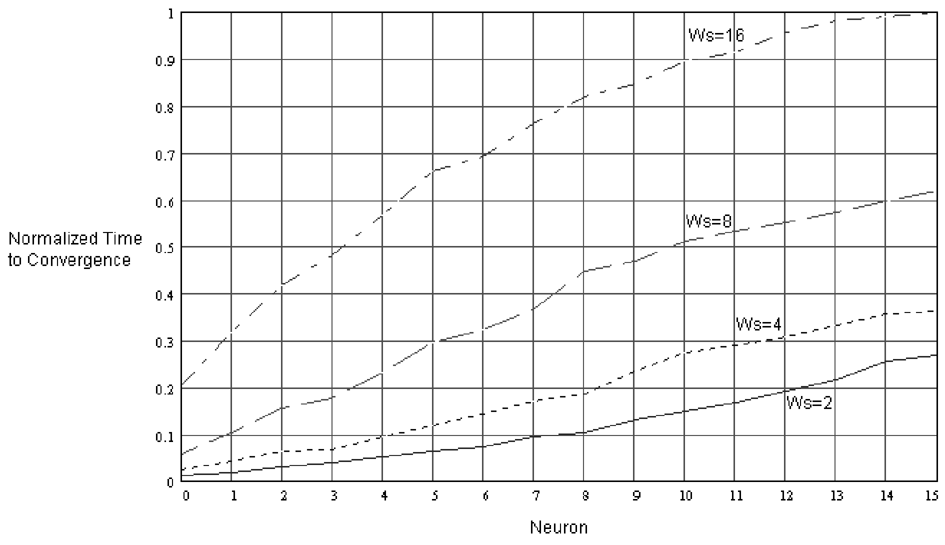

Figure 2: Convergence elapsed time normalized to the maximum value. W_s is the training window size. Training set obtained from Figure 4(a) with $l = 8$ and $n_s = 256$. ANN: $m = 16$, $p = 64$, $\alpha = 1000$ and initial $\eta = 1 \times 10^{-3}$.

window. This condition results from equation (6).

b. The training window size W_s does not affect significantly the number of epochs necessary to the neurons convergence.

As we will see in Section 5, these are not unrealistic assumptions.

The most significant computational cost of the training is incurred in floating point operations. Therefore we will estimate the number of elementary floating point operations O, i.e., the number of sums and products, necessary to the convergence of all neurons [3].

The total number of operations O_{TW} necessary to train the ANN with the GHA-TWA, from equations (4) and (5), is giving by:

$$\begin{aligned} O_{TW} &= \epsilon \left[\frac{-1}{3} {W_s}^3 - (p + \frac{1}{2}) {W_s}^2 \right] \\ &\quad + \epsilon W_s \left[m^2 + 2(p+1)m + \frac{5}{6} + p \right] \end{aligned} \quad (12)$$

The total number of operations O_{GHA} necessary to train the ANN with the single GHA is obtained from (12) with $W_s = m$:

$$O_{GHA} = \epsilon \left[\frac{2}{3} m^3 + (p + \frac{3}{2}) m^2 + (\frac{5}{6} + p) m \right] \quad (13)$$

Equation (14) defines the complexity reduction γ as the ratio of the GHA-TWA com-

$$\gamma = \frac{O_{TW}}{O_{GHA}} \quad (14)$$

Figure 1 shows the complexity reduction γ for $m = 16$ and $p = 64$.

5 Experimental Results

In this section we present some experimental results to assess the TWA performance. The training set is obtained from the image in Figure 4(a) with $l = 8$ and $n_s = 256$. Figures 4(b) and 4(c) show the decompressed image using $W_s = 2$ and $W_s = 16$ (without TWA). The Peak Signal to Noise Ratio (PSNR) between two images $f(x, y)$ and $\widehat{f}(x, y)$ is defined by equation (15).

$$PSNR = 10 log_{10} \left(\frac{255^2}{MSE} \right) \quad (15)$$

where the Mean Square Error (MSE is defined as

$$MSE = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \left(\widehat{f}(x, y) - f(x, y) \right)^2 \quad (16)$$

The PSNR is a similarity measure between images. Notice that the PSNR of images in Figure 4(b) and Figure 4(c) with respect to the orig-

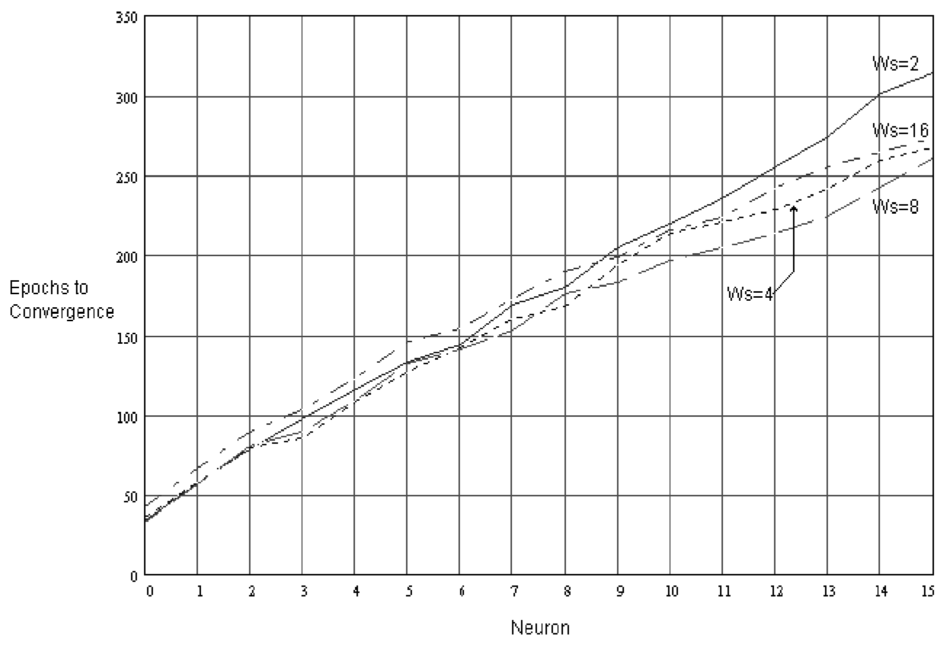

Figure 3: Number of epochs to convergence. W_s is the training window size. Training set obtained from Figure 4(a) with $l = 8$ and $n_s = 256$. ANN: $m = 16$, $p = 64$, $\alpha = 1000$ and initial $\eta = 1 \times 10^{-3}$

that the TWA does not affect the precision of the obtained PCs.

Figure 2 shows the convergence time to each neuron normalized to the maximum value (without TWA). We can verify that the normalized convergence time to neuron 15 for $W_s = 2$, $W_s = 4$ and $W_s = 8$ are quite close to the respective theoretical γ values in Figure 1.

With $W_s = 2$, we can see from Figure 1 that the theoretical complexity reduction is approximately 0.22 and from Figure 2 we can verify that the experimental complexity reduction is approximately 0.27. Thus the GHA complexity was reduced approximately by a factor of 4 with no degradation of the reconstructed image PSNR.

Because the learning rate is adjusted by equation (6) at the end of each epoch and due to the random synapses initialization values, the use of $W_s = 1$ is not feasible. With $W_s = 1$, after the convergence of the neuron N_c, the initial learning rate imposed to the next neuron N_c depends on an eigenvalue obtained from random synapses values. Depending on these synapses values there will be floating point overflow due to a high learning rate. That does not happen with $W_s > 1$ because all neurons subsequent to N_c in the training window have their synapse values already updated. Therefore, the set of synapses value.

Figure 3 shows the number of epochs to the convergence of each neuron for $W_s = 2$, $W_s = 4$, $W_s = 8$ and $W_s = 16$. Notice that, as assumed in Section 2, the TWA does not affect significantly this parameter and that the number of epochs per neuron is nearly constant.

6 Conclusion

In this work we presented the Training Window Algorithm as a proposal to reduce the computational complexity of the General Hebbian Algorithm. This training method reduces the GHA global computational complexity without any degradation of the extracted PCs. Future work will include an attempt to store the samples of the deflation term in equation (4) in an auxiliary matrix, in order to avoid its repeated recomputation.

References

[1] R. Bronson. *Matrix Methods: An Introduc-*

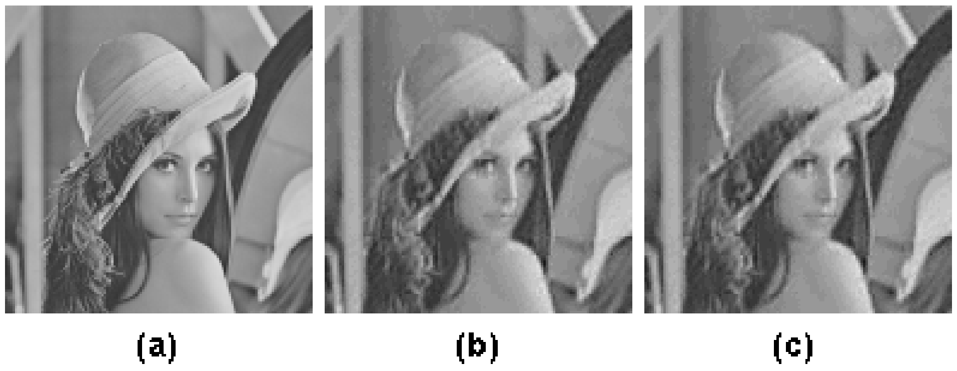

Figure 4: 128×128 pixels 256 gray levels images. Figure 4(b) is the re-expanded original image of Figure 4(a) previously compressed with 16 PCs and with $W_s = 2$. Figure 4(c) is the re-expanded original image of Figure 4(a) previously compressed with 16 PCs and without TWA. Both images present $PSNR = 31.5dB$.

[2] L. H. Chen and S. Chang. An adaptive learning algorithm for principal component analysis. *IEEE Trans. Neural Net.*, 6(5), 1995.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 1991.

[4] S. Bannour e M. R. Azimi-Sadjadi. Principal component extraction using recursive least squares learning. *IEEE Trans. Neural Net.*, 6(2), 1995.

[5] R. Gonzalez and R. E. Woods. *Digital Image Processing.* Addison Wesley, 1993.

[6] M. H. Hassoun. *Fudamentals of Artificial Neural Networks.* MIT Press, 1995.

[7] S. Haykin. *Neural Networks.* Macmillan College, New York, NY, 1994.

[8] L. Xu and A. L. Yulle. Robust principal component analysis by self-organizing rules based on statistical physics approach. *IEEE Trans. Neural Net.*, 6(6), 1995.