ALGORITHMS AND IMPLEMENTATION ARCHITECTURES FOR HEBBIAN NEURAL NETWORKS

J. Andrés Berzal and Pedro J. Zufiria

Grupo de Redes Neuronales
Dpto. de Matemática Aplicada a las Tecnologías de la Información
E.T.S. Ingenieros de Telecomunicación, Universidad Politécnica de Madrid
Ciudad Universitaria S/N, E-28040 Madrid, SPAIN

{abf,pzz}@mat.upm.es

Abstract. Systolic architectures for Sanger and Rubner Neural Networks (NNs) are proposed, and the local stability of their learning rules is taken into account based on the indirect Lyapunov method. In addition, these learning rules are improved for applications based on Principal Component Analysis (PCA). The local stability analysis and the systolic architectures for Sanger NN and Rubner NN are presented in a common framework.

1 Introduction

This paper presents a type of systolic architecture for two types of hebbian NNs, Sanger NN and Rubner NN (with linearized hebbian training approach for direct connections) [1, 2, 4]. The proposed NNs and their architectures implement an on-line PCA from a sample sequence of a stationary vector stochastic process, their weights converging to the eigenvectors of the autocorrelation matrix associated with the input process. The main eigenvector, related to the largest eigenvalue, is approximated by the weights of the first neuron, and so on for the remaining eigenvectors. Hence, PCA finds the uncorrelated directions of maximum variance in the data space, as well as providing the optimal linear projection in the least square sense. PCA is a basic procedure for implementing the Karhunen-Loeve Transform (KLT), widely employed in signal processing systems. Since these NNs implement the KLT from the input data directly, they can perform real time signal processing tasks without explicit computation of the autocorrelation matrix, as well its eigenvalues and eigenvectors (image coding, component analysis of multispectral images, etc) [1, 2, 5, 7].

The implementations of hebbian NNs with a single processor have speed limitations, so they are not appropriate for real time applications being very demanding in computation, high volume data, etc. On the other hand, the systolic architectures with two or more processors are an alternative to the single processor implementations providing parallel processing, modular hardware implementation for

VLSI, etc [4]. In fact, the architectures introduced by this paper are a practical hardware alternative for the real time adaptive calculation of the PCA.

In section 2, Sanger NN and Rubner NN and their training algorithms are presented. In section 3, the local stability of these algorithms is enunciated taking into account the demonstration based on the indirect Lyapunov method. This analysis is appropriate for practical numerical or hardware implementations, where NNs are considered as discrete time dynamical systems. In section 4, these training algorithms are improved extending the PCA to the covariance matrix besides the autocorrelation matrix, Adaptive Learning Rate (ALR), etc. In sections 5 and 6, for both Sanger NN and linearized Rubner NN, some new systolic architectures are presented for hardware implementation. Concluding remarks are indicated in section 7.

2 Sanger NN and Rubner NN Learning Rules and the KLT

The Sanger NN structure and its learning rule, for M neurons with N inputs, are defined by:

$$y_i = \sum_{j=1}^{N} w_{ij} x_j \ , \qquad 1 \le i \le M, \ 1 \le M \le N \ , \tag{1}$$

$$w_{ij,n+1} = w_{ij,n} + \eta_i y_i \left(x_j - \sum_{k=1}^{i} y_k w_{kj,n} \right), \qquad \eta_i \in \Re \ , \tag{2}$$

where y_i, x_j, w_{ij} and η_i are the output, input, weights of the net, and learning rate, respectively. Direct weights are obtained by Sanger learning rule, a modified hebbian learning rule, since it incorporates a second term (Eq. 2). This learning rule leads to the weight vectors to approach the eigenvectors of the autocorrelation matrix of the input samples (patterns), $\mathbf{C}$=E[$\mathbf{xx}^T$].

On the other hand, the Rubner NN structure and its learning rule, for M neurons with N inputs, are defined by:

$$y_i = \sum_{j=1}^{N} w_{ij} x_j + \sum_{j=1}^{j<i} u_{ij} y_j \ , \ 1 \le i \le M, \ 1 \le M \le N \ , \tag{3}$$

$$w_{ij,n+1} = \frac{w_{ij,n} + \eta_i y_{i,n} x_{j,n}}{\left| \mathbf{w}_{ij,n} + \eta_i y_{i,n} \mathbf{x}_{j,n} \right|}, \quad u_{ij,n+1} = u_{ij,n} - \gamma_i y_{i,n} y_{j,n}, \ \ \eta_i, \gamma_i \in \Re \ , \tag{4}$$

where y_i represents the output, x_j is an input, w_{ij} are the direct weights, u_{ij} are the lateral weights between y_j and y_i, and η_i and γ_i are the learning rates for the direct and lateral weights, respectively. The direct weights are adjusted upon normalized hebbian learning and the lateral weights, upon antihebbian learning (Eq. 4). The normalization of the direct weights of the Rubner learning rule can be avoided for small η_i, so that this learning rule for w_{ij} can be approximated by the linear term of the

Taylor expansion [2] (Eq. 5). Also, this linear formulation reduces the complexity of the learning rule in terms of operations (processing time).
For Rubner NN and its linearized version, the direct weights converge to the eigenvectors of the autocorrelation matrix of the input samples (patterns), **C**=E[**xx**^{T}], and the lateral weights converge to the null vector, **0**.

$$w_{ij,n+1} = w_{ij,n} + \eta_i y_{i,n}\left(x_{j,n} - (\mathbf{w}_{i,n}^T \mathbf{x}_n) w_{ij,n}\right), \qquad \eta_i \in \Re . \tag{5}$$

3 Local Stability for Sanger NN and Rubner NN Learning Rules

The local stability of the Sanger and linearized Rubner learning rules is analytically proved via the indirect Lyapunov method [2]. Other studies to prove the local stability of Sanger and Rubner learning rules are based on their associated ODE (Ordinary Differential Equation) [4, 6]. However, in this paper the local stability study used maintains its discrete time evolution. Therefore, it allows to obtain the local stability conditions taking into account the learning rates.

For the indirect Lyapunov method, the Sanger and linearized Rubner learning rules are expressed in terms of the weights, and the mean operator is applied to both sides of the learning equations which are transformed from the deterministic framework to the probabilistic one. Their fixed points are obtained assuming that input samples (**x**) and weights ($\mathbf{w}_i$ and $\mathbf{u}_i$ only for linearized Rubner learning rule) are statistically independent, and that there are different scales of time for the evolution of each weight. For Sanger, its fixed points take the form of vector $\pm\mathbf{e}_j$ ($i \leq j$) and for Rubner, its fixed points are vector pairs **(**$\pm\mathbf{e}_j$, **0)** ($i \leq j$), where $\mathbf{e}_j$ is an eigenvector of **C**=E[**xx**^{T}]. Applying Lyapunov method, for Sanger learning rule, the asymptotically stable fixed points are $\mathbf{w}_i=\pm\mathbf{e}_i$ if $\eta_i<1/\lambda_i$ and for linearized Rubner learning rule, they are **(**$\mathbf{w}_i$, $\mathbf{u}_i$**)=(**$\pm\mathbf{e}_i$, **0)** if $\eta_i<1/\lambda_i$ (assuming $\gamma_i>>2\eta_i$).

4 Improvements for Sanger NN and Rubner NN

The performance of Sanger and Rubner learning algorithms can be improved [1, 3]:

– The direct weights can converge to the eigenvectors of the covariance matrix of the input samples. This is accomplished by a dynamic computation of the time mean value of the input samples $\bar{\mathbf{x}}_n$ (Eq. 6) and subtracting this value to the input to the NN. $\bar{\mathbf{x}}_n$ converges to E[**x**] in probability, assuming $\mathbf{x}_n$ are uniform random variables with finite variance.

$$\bar{\mathbf{x}}_n = \bar{\mathbf{x}}_{n-1} + \frac{\mathbf{x}_n - \bar{\mathbf{x}}_{n-1}}{n}, \quad \bar{\mathbf{x}}_1 = \mathbf{x}_1 . \tag{6}$$

– The dynamic evaluation of the eigenvalues $\tilde{\lambda}_{in}$ (Eq. 7) of the autocorrelation or covariance matrix of the input samples, whose magnitude and location (neuron i)

provides information about the importance and the convergence of the weights (eigenvectors). $\tilde{\lambda}_{i,n}$ converges to E[y_i^2]=λ_i in probability, being y_i^2 uniform random variables with finite variance (assuming the weights of the NN converge to their fixed points).

$$\tilde{\lambda}_{i,n} = \tilde{\lambda}_{i,n-1} + \frac{y_{i,n}^2 - \tilde{\lambda}_{i,n-1}}{n}, \qquad \tilde{\lambda}_{i,n} = y_{i,1}^2, \;\; \tilde{\lambda}_{i,n} = \overline{y_i^2}_{\;n} \, . \tag{7}$$

– The Adaptive Learning Rate (ALR) defined by Eq. 8 for each neuron accelerates and synchronises the training phase. This is only useful for the Sanger NN, since for the Rubner NN it amplifies the oscillation in each component of the weights [1]. The experimental simulations of Sanger and Rubner learning algorithms have proved that the ALR does not modify the convergence properties (fixed points and local stability) with respect to their initial algorithms (section 3).

$$\eta_{i,n} = \eta_0 \frac{\tilde{\lambda}_{1,n}}{\tilde{\lambda}_{i,n}}, \qquad \tilde{\lambda}_{i,n} \neq 0 \, . \tag{8}$$

5 Systolic Architectures for Hebbian Neural Networks

In this section, we propose some new systolic architectures for the Sanger NN and the Rubner NN which implement the retrieving phase (input and output function of the NN) and training phase for both NN. The graphical representation of the systolic architecture is given by the Dependence Graph (DG). It shows the topology of the network, nodes, dependencies (input and output), flow of data, etc, and all of them define the function to implement. The systolic architecture is suitable for hebbian NNs and it is a 2-D array (Fig. 1). The number of rows and columns correspond to the number of output and inputs of the NN respectively. The operations of each node are basically multiply-accumulate and commutation.

The projection of the DGs onto a linear array of processors defines the architecture. The vertical projection for the hebbian NNs, where each processor executes a row of the DG (neuron) is selected in this paper since it gives more simplicity than others (Fig. 2). The number of cycles to process one input sample for the Sanger NN linear arrays for both retrieving and training architectures is $N+M-1$, less than NM cycles required by classical architectures (one processor). For linearized Rubner NN architectures this situation also happens, namely the number of cycles for their linear arrays for both retrieving and training architectures is $N+2M-1$, less than $NM+(M-1)M/2$ cycles required by classical architectures. The interconnection between retrieving and training linear arrays for linerized Rubner NN requires a synchronization for input/output data, unnecessary for Sanger NN.

The processors, flow of data, etc, for these linear arrays are different and mainly they depend on the NN and the function to implement (retrieving or training):

– **Retrieving Linear Array** of **Sanger NN** (Fig. 2). Its processor *j* has as serial inputs the weights of the neuron *j* (**w**$_j$) stored in a circular register and the components of the input samples (**x**). After *N* cycles this processor provides the output of neuron *j* (y_j). The processor *j*+1 receives **x** in serial with a delay of one cycle; then y_{j+1} is obtained one cycle later than y_j. The cycle for these processors is determined mainly by the time needed to execute one multiplication and one addition in serial.

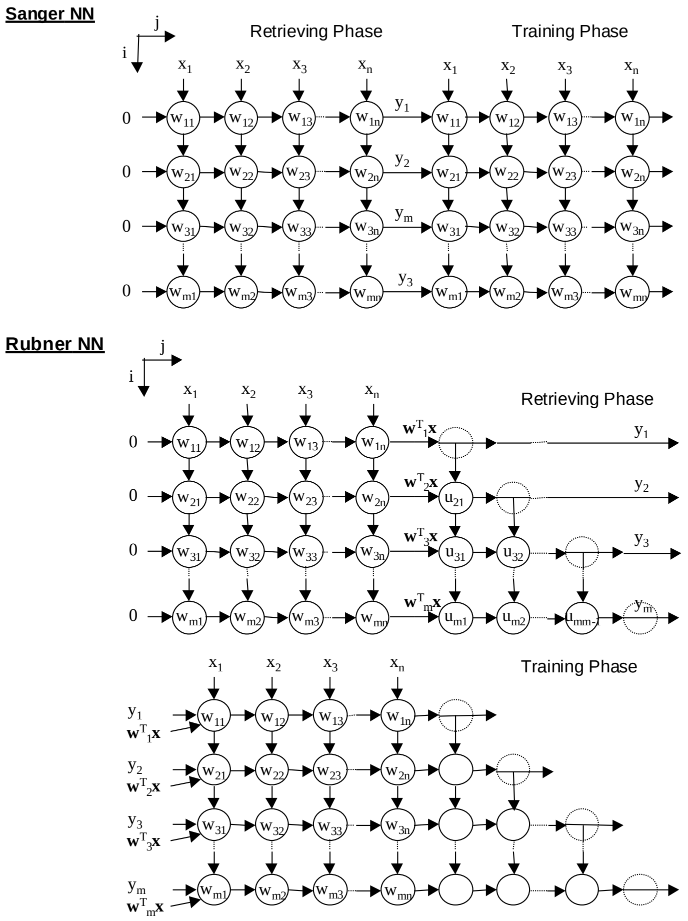
Sanger NN
Retrieving Phase
Training Phase
Rubner NN
Retrieving Phase
Training Phase
j
i
x_1 x_2 x_3 x_n
0
w_{11} w_{12} w_{13} w_{1n}
w_{21} w_{22} w_{23} w_{2n}
w_{31} w_{32} w_{33} w_{3n}
w_{m1} w_{m2} w_{m3} w_{mn}
y_1 y_2 y_m y_3
$\mathbf{w}^T_1\mathbf{x}$ $\mathbf{w}^T_2\mathbf{x}$ $\mathbf{w}^T_3\mathbf{x}$ $\mathbf{w}^T_m\mathbf{x}$
u_{21} u_{31} u_{32} u_{m1} u_{m2} u_{mm-1}

Fig. 1. Dependence Graphs (DGs) of a systolic architectures for Sanger NN and Rubner NN.

– **Retrieving Linear Array** of **linearized Rubner NN** (Fig. 2). Its processor *j* differs from the processor *j* of the retrieving linear array of the Sanger NN in several aspects for the processor *j*, its computing time to provide y_j is *N*+*j*-1 cycles, the direct and lateral weights ($\mathbf{w}_j$ and $\mathbf{u}_j$) are stored in a circular register (first $\mathbf{w}_j$), a new output for the scalar product $\mathbf{w}^T{}_j\mathbf{x}$ is incorporated, and processor *j* sends to processor *j*+1 **x** and y_k ($0<k<j$) in serial. The cycle for this processor is determined mainly by the time needed to execute two multiplications and one addition in serial.

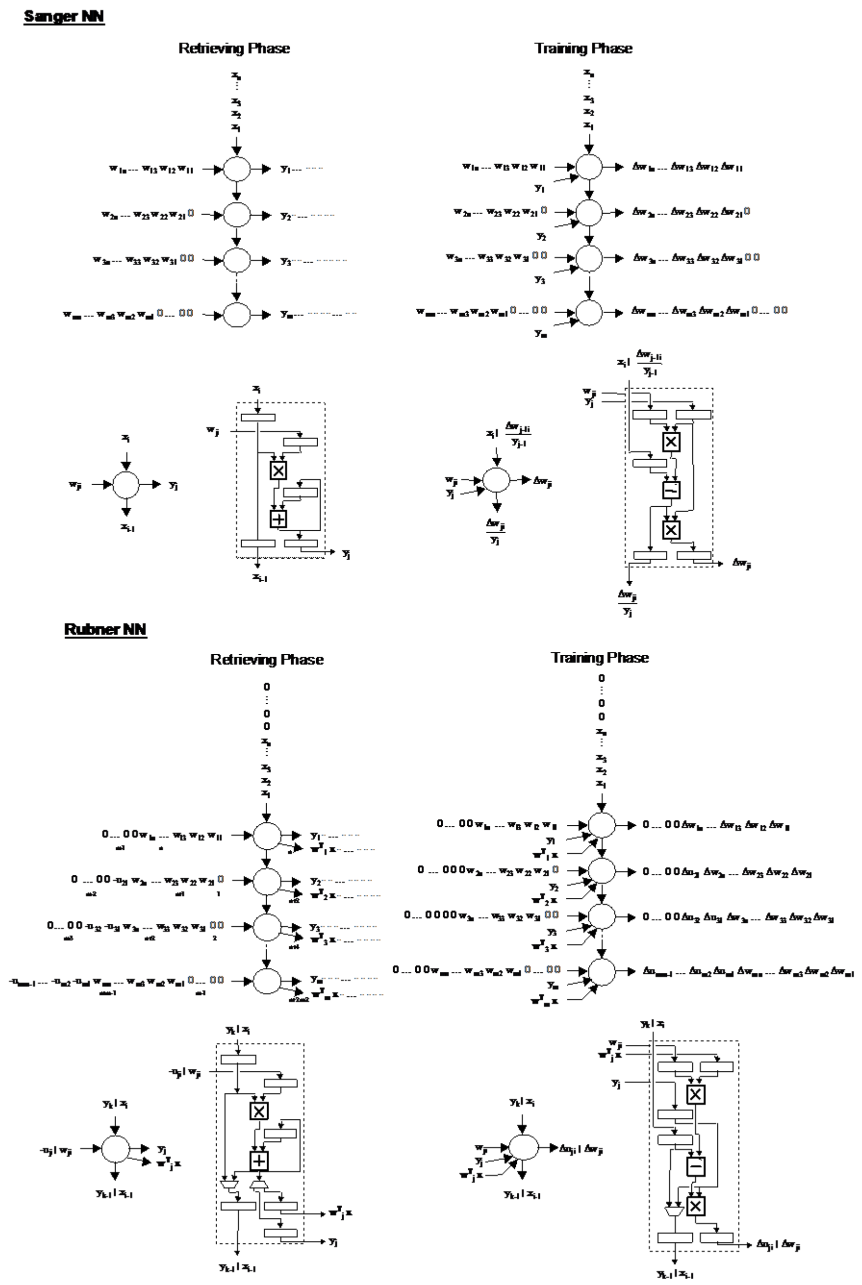

Fig. 2. Linear arrays of processors (vertical projection of the DG) and their processor schemes of a systolic architecture for Sanger NN and Rubner NN (retrieving and training phases).

– **Training Linear Array** of **Sanger NN** (Fig. 2). Its processor *j* has as inputs the weights of the neuron *j* ($\mathbf{w}_j$) stored in a circular register, its associated output (y_j)

and the components of the weight increments of the previous neuron normalized by its output ($\Delta\mathbf{w}_{j\text{-}1}/\mathrm{y}_{j\text{-}1}$), except for the first processor where we have the components of the input samples (**x**). After N cycles this processor provides the weight increment $\Delta\mathbf{w}_j$ (update $\mathbf{w}_j$ next step). The processor j+1 receives $\Delta\mathbf{w}_j/\mathrm{y}_j$ in serial with a delay of one cycle then $\Delta\mathbf{w}_{j+1}$ is obtained one cycle later than $\Delta\mathbf{w}_j$. The cycle for these processors is determined mainly by the time needed to execute two multiplications and one subtraction in serial.

– **Training Linear Array** of **linearized Rubner NN** (Fig. 2). Its processor j differs from the processor j of the training linear array of the Sanger NN in the following aspects: the computing time to provide $\Delta\mathbf{w}_j$ and $\Delta\mathbf{u}_j$ for the neuron j are N+j-1 (update $\mathbf{w}_j$ and $\mathbf{u}_j$ next step), the direct and null weights ($\mathbf{w}_j$ and **0**) are stored in a circular register (first $\mathbf{w}_j$), a new input for the scalar product $\mathbf{w}^{\mathrm{T}}{}_j\mathbf{x}$ is incorporated, and the processor j sends to processor j+1 **x** and and y_k ($0<k<j$) in serial. The cycle for this processor is determined mainly by the time needed to execute two multiplications and one subtraction in serial. This linear array is valid for APEX NN [4] if the null weights are replaced by $\mathbf{u}_j$.

For the training phase there is an alternative to the Data Adaptive (DA) scheme described in the previous points, weights being updated before the next input pattern is processed (Fig. 2). This is the Block Adaptive (BA) scheme, the weight update being postponed until the end of each training data block. Note that the BA scheme requires less computation than the DA scheme since the weight updating is done less frequently according to the size of the blocks. On the other hand, the size of the blocks should affect minimally to the convergence speed versus the computing time.

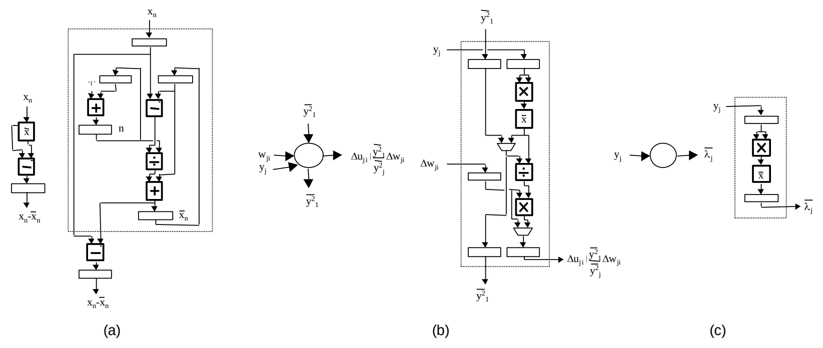

Fig. 3. Improving the features of the processors of the systolic architectures for Sanger NN and Rubner NN: (a) PCA with the covariance matrix, (b) ALR and (c) eigenvalue calculation.

6 Improvements for the Hebbian NN Architectures

The improvements for the Sanger NN and Rubner NN defined in section 4 can also be implemented by the systolic linear array architectures.

These linear architectures can be adapted to support PCA with covariance matrix if the input for input samples incorporates a preprocessing input block. This block should have a dynamic time mean unit (Fig. 3.a) with circular registers (memory for each component) synchronized with the components of the input samples, then the input for the Sanger NN or Rubner NN is the output of the dynamic time mean unit minus the initial input. The cycle of this block is the time needed for two subtractions, one sum and one division in serial. As well, it is possible that these linear array architectures use ALR. Then, the weight increment outputs of each processor are treated by other processor which modulates the increments according to Eq. 8 (Fig. 3.b). The cycle of this processor is the time needed for two multiplications, the time for one step of the dynamic mean value unit and one division in serial. The eigenvalue associated with each neuron can be calculated using a processor for each neuron whose input is the output of its associated neuron (retrieving array processor) and the output is the time mean value of its square input which converges to the eigenvalue of its associated neuron (Fig. 3.c). The cycle of this processor is the time for one multiplication and the time for one step of the dynamic mean value unit.

7 Concluding Remarks

The present work has focused on the study of Sanger NN and Rubner NN, as well as for Rubner NN its linearized formulation, and a new systolic architecture for their implementation. First of all, the presented NN has been introduced from their empirical formulation, some improvements of their algorithms have been addressed and their local stability has been enunciated. Finally, a systolic architecture for these NNs is described using its DG and projected on a linear array of processors including the improvements for their algorithms.

References

1. Berzal, J.A., Zufiria, P.J. and Rodríguez, L.: Implementing the Karhunen-Loeve Transform via Improved Neural Networks. Proceedings of the International Conference on Engineering Applications of Neural Networks, London 15-17 June 1996, 375-378.
2. Berzal, J.A. and Zufiria, P.J.: Linearized Trainning of Hebbian Neural Networks, Aplication to Multispectral Image Processing. Proceedings of the International Conference on Engineering Applications of Neural Networks, Gibraltar 10-12 June 1998, 1-8.
3. Chen, L.H. and Chang, S.: An adaptive Learning Algorithm for Principal Component Analysis. IEEE, Transactions on Neural Networks, September 1995, 1255-1263.
4. Diamantaras K.I. and Kung, S. Y.: Principal Component Neural Networks, Theory and Applications. John Wiley & Sons, Inc, 1994.
5. Dony, R.D. and Haykin, S.: Neural Networks Approaches to Image Compression. Proceedings of the IEEE, February 1995, 288-303.
6. Weingessel, A. and Hornik, K.: Local PCA Algorithms. IEEE, Transactions on Neural Networks, November 2000, 1242-1250.
7. Zufiria, P.J., Berzal, J.A., Martínez, M.A. and Fernández Serdán, J.M.: Neural Network Processing of Satellite Data for the Nowcasting and Very Short Range Forecasting. Proceedings of the International Conference on Engineering Applications of Neural Networks, Warsaw, 13-15 September 1999, 241-246.