In: J.Delgado-Frias, W.Moore (Eds.): VLSI for Artificial intelligence and Neural Networks, Plenum Press, New York, 1994, pp.53-60

A VLSI-DESIGN OF THE MINIMUM ENTROPY NEURON

R. Brause
J.W. Goethe-University, Frankfurt a. M., Germany
brause@informatik.uni-frankfurt.de

Abstract

One of the most interesting domains of feedforward networks is the processing of sensor signals. There do exist some networks which extract most of the information by implementing the maximum entropy principle for Gaussian sources. This is done by transforming input patterns to the base of eigenvectors of the input autocorrelation matrix with the biggest eigenvalues. The basic building block of these networks is the linear neuron, learning with the Oja learning rule.

Nevertheless, some researchers in pattern recognition theory claim that for pattern recognition and classification clustering transformations are needed which reduce the intra-class entropy. This leads to stable, reliable features and is implemented for Gaussian sources by a linear transformation using the eigenvectors with the *smallest* eigenvalues.

In another paper (Brause 1992) it is shown that the basic building block for such a transformation can be implemented by a linear neuron using an Anti-Hebb rule and restricted weights. This paper shows the analog VLSI design for such a building block, using standard modules of multiplication and addition.

The most tedious problem in this VLSI-application is the design of an analog vector normalization circuitry. It can be shown that the standard approaches of weight summation will not give the convergence to the eigenvectors for a proper feature transformation. To avoid this problem, our design differs significantly from the standard approaches by computing the real Euclidean norm.

Keywords: minimum entropy, principal component analysis, VLSI, neural networks, surface approximation, cluster transformation, weight normalization circuit.

INTRODUCTION

For many purposes the necessary processing of sensor input signals is realized by using a system which implements the maximization of the transinformation from the input to the output of the system. For deterministic systems, this corresponds to the maximization of the output entropy (maximum entropy principle). In pattern recognition theory, it is well known that for Gaussian distributed sources this corresponds to the minimization of the mean square error of the output. For linear systems, this is done by a linear transformation to the base of the eigenvectors of the autocorrelation matrix (Fukunaka 1972). Furthermore, we can compress (encode) the input information by using only the base vectors (eigenvectors) with the biggest eigenvalues. Neglecting the ones with the smallest eigenvalues ($m<n$, see Fig.1) results in the smallest reconstruction error on the encoded input (Fukunaga 1972). Generally, this approach can be used for sensor signal coding such as picture encoding, see e.g. (Jayant and Noll 1984).

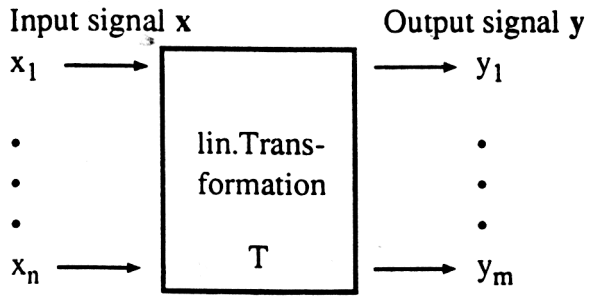


Figure 1 The feature signal information

The neural network models of this approach use linear neurons, where each neural weight vector corresponds to one eigenvector. Examples of those architectures are the Oja subspace network (Oja 1989, Williams 1985), the Sanger decomposition network (Sanger 1989) and the lateral inhibition network of Földiák (1989) or Rubner and Tavan (1989). The first mentioned networks decompose sequentially the input vector **x**, see figure 2. They use as a basic building block the linear correlation neuron which learns the input weights by a Hebb-rule, restricting the weights $\mathbf{w}_1 \ldots \mathbf{w}_n$.

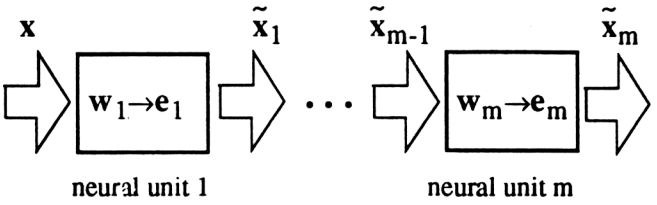


Figure 2 The sequential learning of the eigenvectors

The input for each stage is obtained by subtracting sequentially all the projections of the weight vectors on the input pattern

$$\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}_{i-1} - y_i \mathbf{w}_i \qquad \tilde{\mathbf{x}}_0 = \mathbf{x}$$

As Oja showed (Oja 1982), this learning rule let the weight vector of the first neuron convergence to the eigenvector of the expected autocorrelation matrix $\mathbf{C}$ of the input patterns $\tilde{\mathbf{x}}$ with the biggest eigenvalue λ_{max}:

$$\mathbf{w}_1 \to \mathbf{e}_k \quad \text{with } \lambda_k = \max_i \lambda_i \text{ and } \mathbf{C}_i\mathbf{e}_i = \lambda_i\mathbf{e}_i \text{ with } \mathbf{C}_i = \left\langle \tilde{\mathbf{x}}_{i-1}\tilde{\mathbf{x}}_{i-1}^T \right\rangle$$

The subsequent neuron learns the eigenvector with the biggest eigenvalue of the autocorrelation matrix of the remaining input: The eigenvectors remain the same as before, but in this system λ_k becomes zero and the second eigenvalue becomes in fact the biggest one.

The whole network is used in two modes: in the learning mode, where the input is propagated sequentially through the units, and in the transforming, filtering mode, where the input is presented to the neurons in parallel, see figure 3. The basic building block of these networks is the linear neuron, learning with the Oja learning rule (Oja 1982, Sanger 1989); the same as in other asymmetric networks with additional lateral inhibition (Rubner and Tavan 1989, Földiák 1989).

Nevertheless, some researchers in pattern recognition theory (Tou and Gonzales 1974) claim that for pattern recognition and classification clustering transformations are needed which reduce the intra-

class entropy. This leads to stable, reliable features and is implemented for Gaussian sources by a linear transformation using the eigenvectors with the *smallest* eigenvalues.

Additionally, the well-known, important problem of approximating randomized, disturbed measures of hypersurfaces can be solved by minimizing the total least square error (TLSE) of an approximation surface. It can be shown that the approximated surface coefficients can be obtained by the minor components (eigenvectors with the smallest eigenvalues) of the observed data (Xu,Oja and Suen 1992).

In another paper (Brause 1992) it is shown that the basic building block for such a transformation can be implemented by a linear neuron using an Anti-Hebb rule and restricted weights. More formally, the transformation of *n* input variables $(x_1,..,x_n) = \mathbf{x}$ to *m* output variables $(y_1,..,y_m) = \mathbf{y}$ is made by a linear transformation. For one output variable y_i the transformation can be implemented by a formal neuron. For each neuron, the input is weighted by the weights $\mathbf{w} = (w_1,..,w_n)$ and summed up to the activation z of the neuron

$$y_i = z(t) = \sum_{j=1}^{n} w_j x_j = \mathbf{w}^T \mathbf{x} \qquad \textit{linear activation} \qquad (1)$$

which is expressed as the scalar product of **x** and the transpose of **w**. Additionally, the transformation coefficients w_j are locally learned on chip by the stochastic version of the Hebbian law

$$\hat{\mathbf{w}}(t+1) = \mathbf{w}(t) - \gamma(t)\, \mathbf{x}(t)\, y(t) \qquad \textit{Anti-Hebb-Rule} \qquad (2)$$

$$\text{and } \mathbf{w}(t+1) = \frac{\hat{\mathbf{w}}(t+1)}{\sqrt{\sum_j \hat{w}_j(t+1)^2}} \qquad \textit{Normalization} \qquad (3)$$

This paper shows the VLSI design for such a building block, using standard modules of multiplication and addition for Eqs(l) and (2). Additionally, the normalization of Eq. (3) is covered by special circuitry, described in the last section.

THE SYSTEM DESIGN

One of the most interesting objectives for a VLSI design is a fast, real-time oriented architecture design. The neural network paradigm of small, parallel processors allows an efficient, parallel implementation of the proposed transformation. For such a system, the layout in figure 3 shows how the input vector of *n* sensor lines is fed in parallel to all *m* neurons via an input bus.

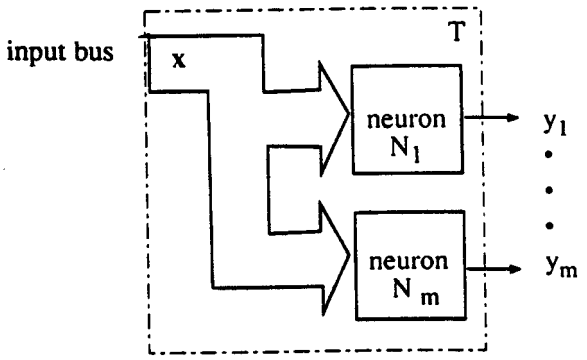


Figure 3 Signal transformation by formal neurons

Each neuron performs a projection of the input to a weight vector by equation (1). The output bus consists of the *m* output lines of the *m* neurons.
Additionally, the weights w_{ij} are updated according to Eqs. (2) and (3) in such a way that they will converge to the eigenvector of the autocorrelation matrix with the smallest eigenvalue. This is reflected by the principal system design for one neuron in figure 4.

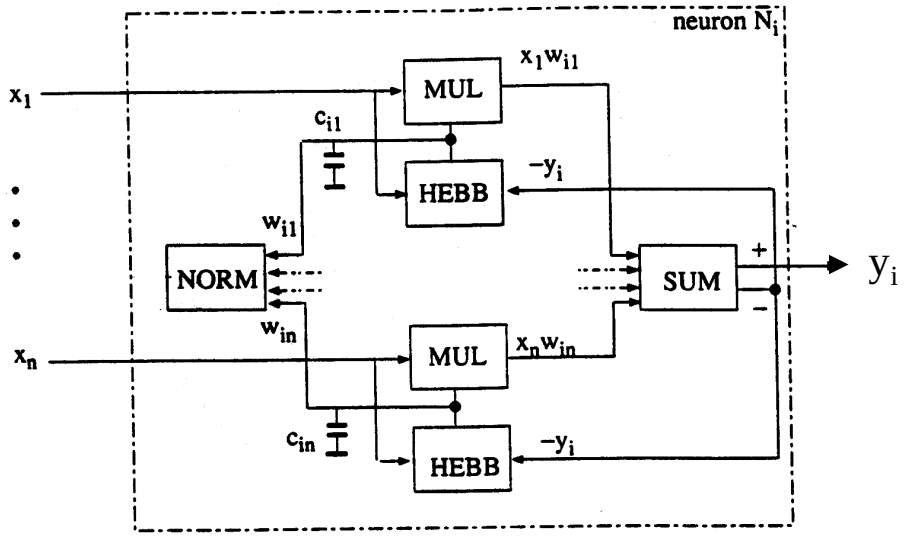


Figure 4 The principal system structure of one neuron

Nevertheless, to achieve a complete eigenvector decomposition necessarily an interaction between the neurons has to exist. There are several possibilities to construct such a network, see Brause (1992). Since the interactions consist of simple multiplications and additions (see e.g. first section), they are not shown in figure 3 and we concentrate on the basic implementation features for one neuron.
Modeling the weights by voltages w_{ij} at capacitors c_{ij}, each input signal x_j is first multiplied by the weight w_{ij} in the MUL module and the resulting current is summed up in SUM, generating the output signal after Eq. (1). The learning according to Eq. (2) takes place in module HEBB, whereas the normalization of Eq.(3) is done in NORM.
Note that we do not use EEPROMS, floating gate transistors or other permanent means for the storage of the weight values. Thus, the learning takes place in parallel to the normal signal processing of Eq.(l) and reflects therefore the real statistics of a short time period of the input signal, not the prefixed ones of one application. Therefore, the design is self-adaptive in respect to the application problem statistics. In the case where you do not know the application properties (one chip for all applications) this yields an optimal transformation. Nevertheless, when the statistics are well known the design can be simplified by cancelling the learning mechanism (HEBB and NORM modules) and implementing the well known, constant weights directly.
Let us discuss now the implementation of the various modules.

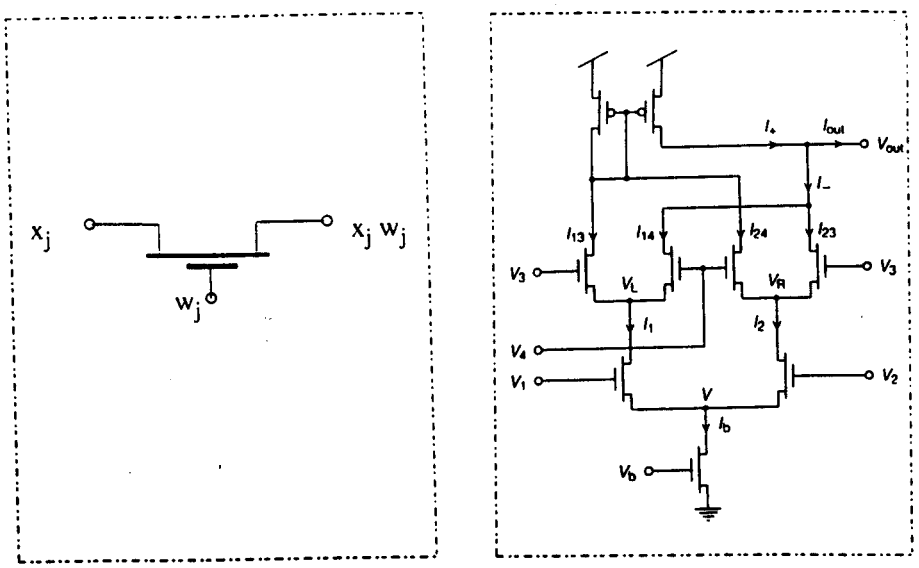
THE BUILDING BLOCKS

There already do exist some "standard" building blocks for the purpose of neural networks which are cited in several papers (Card and Moore 1989, Mead 1989, Vittoz 1989). Nevertheless, we do not

use some of them for certain reasons; others are modified and there are some new building blocks in this paper. In this section, all building blocks are discussed in detail.

The standard building blocks

One of the best known building blocks in figure 4 is the multiplication module MUL. When the input voltage x_j is smaller than the weight voltage w_{ij}, the circuit for positive and negative multiplication might consists of just one FET (linear region) (Weste and Eshraghian 1985, Card and Moore 1989), see figure 5a on the left hand side.



a) one-transistor multiplication **b)** Four quadrant Gilbert multiplier for $(V_{I-V2})X(V3-V4)$

Figure 5 Multiplication circuits

Very often, the input signal is bigger than the weight signal and the transistors leaves the state of weak inversion, resulting in non-linear current. Therefore, we propose the linear region of the well-known Gilbert four-quadrant multiplier circuit (Gilbert 1968, Gilbert 1974) or a wide-range version of it (e.g. Mead 1989) shown in figure 5b on the right hand side for the multiplication $w_{ij}x_j$ in the MUL building block. This is also true for the other multiplication x_jy_i of Eq.(2) in the HEBB building block. Here, we charge the capacitor c_{ij} by a current proportional to the product $w_{ij}x_j$ (see e.g. Card and Moore 1989).

Another important action is the general SUM building block, which exists only once per neuron and implements the sum of Eq.(l). Here we do not use the virtual ground circuit, proposed for instance by Vittoz (1989), because it also automatically scales the input which leads to false results in our case. Instead, we relay on a simple current adder, followed by an amplifier in figure 6.

There remains the most interesting building block, the normalization block which implements Eq.(3). Up to my knowledge, there do not yet exist propositions for an implementation of it. This is covered by the next section.

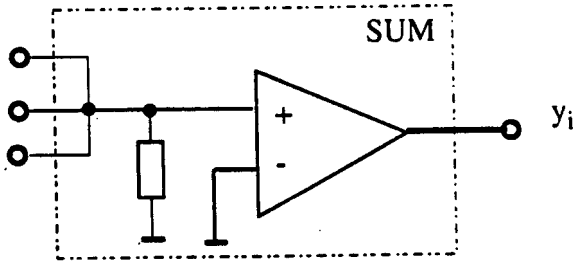


Figure 6 The sum circuit

The normalization block

The basic idea for the mechanism of the learning algorithm used in this paper is the restriction of the resources, the weights. The restriction or side-condition consists of a constant length of the weight vector which is fixed at all iteration steps according to eq. (1.3). To achieve this, the normalization block uses three non-linear circuits.
The most tedious problem in this VLSI-application is the design of this vector normalization circuitry. It can be shown that the standard approaches of weight normalization (Mead 1989, Vittoz 1989, Card and Moore 1989)

$$\mathbf{w}(t+1) = \frac{\hat{\mathbf{w}}(t+1)}{\sum_j \hat{w}_j(t+1)} \qquad (4)$$

will *not* give the convergence to the eigenvectors for a proper feature transformation because the weights are only normalized in the L_1 norm sense. To avoid this problem, our design differs significantly from the standard approaches by computing the L_2 norm, the real quadratic Euclidean norm of Eq.(3).

The weights signal levels are transformed to squared signals (see figure 7) and then, by a current mirror, induce a proportional current in a Kirchhoff network which is supplied by a constant current.

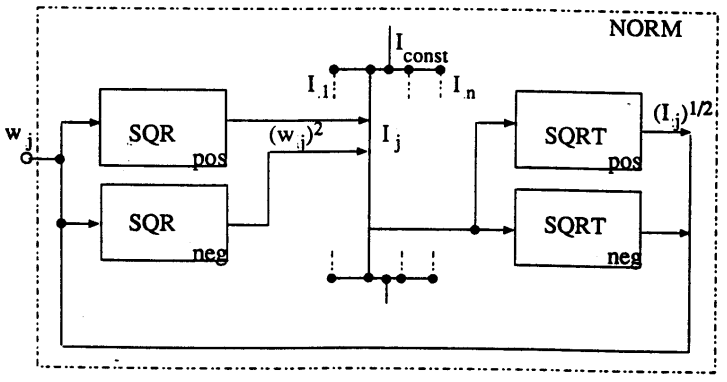


Figure 7 The normalization block schema

The relative currents in all branches will be balanced according to the relative values of the squared input signals, such implementing the normalization relations. The normalized currents are then read out

by a current mirror, squeezed by a root circuit and fed back to the input where they control the factor γ of Eq. (2) by regulating the output resistance of the root amplifier building block. Thus, the weight signals are normalized. More formally, this can be shown as follows.

Assume that for a certain weight vector w all currents of the constant current network have a certain value I_j such that

$$I_{const} = \sum_j I_j \text{ and } I_j \sim w_j^2 \tag{5}$$

Now, let us further assume that in time step t+l each component of **w** has changed independently by a factor of a_j

$$\hat{w}_j(t+1) = a_j w_j(t) \tag{6}$$

The signal $\hat{w}_j(t+1)$ is converted to its square $\hat{w}_j(t+1)^2$ and induces proportionally a conductance $L_j = 1/R_j$ in the branch I_j . With the increase of each conductance

$$L_j(t+l) = a_j^2 L_j(t) \tag{7}$$

the conductance L of all branches becomes

$$L(t+l) = \Sigma_j L_j(t+l) = \Sigma_j a_j^2 L_j(t) = \alpha L(t) \tag{8}$$

$$\text{with } \alpha = (\Sigma_j a_j^2 L_j(t)) / L(t)$$

Since the product of the voltage V and the conductance L, the current source $I_{const} = V(t) \cdot L(t)$, is constant, we know with (8) and

$$I_{const} = V(t+1)\, L(t+1) = V(t+1)\alpha L(t) = V(t)L(t) \tag{9}$$

that the voltage V(t) becomes

$$V(t+l) = V(t)/\alpha \tag{10}$$

By (7) and (10), the current in the j-th branch becomes

$$I_j(t+1) = V(t+1)L_j(t+1) = V(t)a_j^2 L_j(t)/\alpha = I_j(t)a_j^2/\alpha \tag{11}$$

and therefore, the j-th component of the weight vector

$$w_j(t+1)^2 = w_j(t)^2 a_j^2/\alpha \tag{12}$$

becomes scaled by the factor of Eq.(6). Additionally, the whole vector is scaled in length by a factor of α which results in a length of

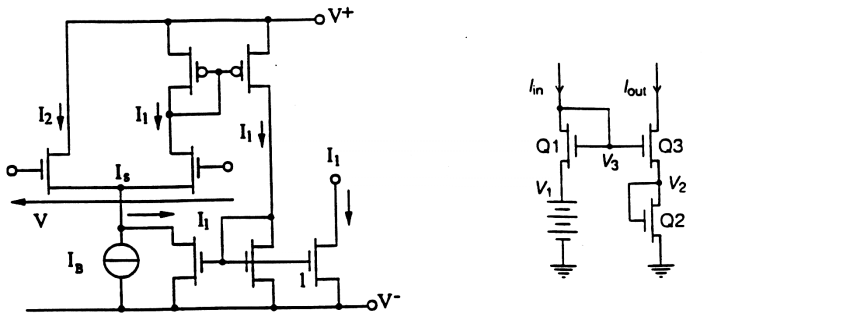
$$\mathbf{w}(t+1)^2 = \sum_j w_j(t+1)^2 = \sum_j w_j(t)^2 a_j^2 / \alpha = \frac{\sum_j w_j(t)^2 a_j^2 L(t)}{\sum_j a_j^2 L_j(t)}$$

and with $L_j(t) = c w_j(t)^2$, $L(t) = \Sigma_j L_j(t) = \Sigma_j c w_j(t)^2 = c\, \mathbf{w}(t)^2$ with a proportional factor *c*

we get $$\mathbf{w}(t+1)^2 = c\,\mathbf{w}(t)^2 \frac{\sum_j w_j(t)^2 a_j^2}{\sum_j a_j^2 c w_j(t)^2} = \mathbf{w}(t)^2$$

Thus, the length of the vector **w** will remain fixed, only determined by the fixed value of the constant current.

The next figures will cover the rest of the non-linear circuits. The nonlinear, over proportional square effect is achieved by a kind of feed-back circuit, originally proposed by Degrauwe et al. (1982), shown in figure 8 on the left hand side.



a) The square circuit (after Vittoz 1989) **b)** the square root circuit (after Mead 1989)

Figure 8 Expanding and compressing circuits

The non-linear square root circuit is covered by a simple current-mirror design, described for instance in Mead (1989) and shown on the right hand side of figure 8.

Nevertheless, for a real VLSI implementation of this approach some analog detail problems must be solved. One of them is the question, how the coupling of the non-linear normalization feed-back block to the Hebb-learning unit should be designed.

On the one hand, when a normalization of the weight signal is obtained, the ratio of the output resistance of the Hebb-circuit to the output resistance of the root circuit must be high to highly influence the weight signal. Obviously, a small error due to the finite ratio will be left.

On the other hand, this error voltage is just the one which can influence the normalization circuit and will change the direction of the weight vector. So, the error should not be made too small. Current research is done to overcome this problem.

CONCLUSION

The paper shows how the linear transformation of multidimensional signal features to the optimal feature set, the base of eigenvectors implemented by formal neurons, can be accomplished. The main contribution of this paper to the analog VLSI design of formal neurons consists in a system concept for such a transformation and an analog vector normalization mechanism proposed for the first time. The desired properties of this mechanism is proven on the macro unit level.

A neuron with such a normalization mechanism can be used in a network either for coding purposes and signal decorrelation (maximum entropy network) or for cluster transformation, classification preprocessing or surface fitting (minimum entropy network).

References

Brause, R., "The Minimum Entropy Neuron- a basic Building Block for Clustering Transformations", in *Artificial Neural Networks 2*, I. Aleksander and J. Taylor (Eds.), Elsevier Sc.Publ., pp. 1095-1098, 1992

Card, H. C. and Moore, W. R., "VLSI Devices and Circuits for Neural Networks", *Int. J. Neural Systems*, Vol 1/2, pp. 149-165, 1989

Degrauwe, M. et al., "Adaptive biasing CMOS amplifiers", *IEEE Journal of Solid-State Circuits*, SC-17, p. 522, 1982

Földiák, P., "Adaptive network for optimal linear feature extraction", in *Proc. IEEE Int. Joint Conf. on Neural Networks*, pp. I 405-405, San Diego 1989, CA

Fukunaga, K., *Introduction to Statistical Pattern Recognition*, Academic Press, New York 1972.

Gilbert, B., "A precise four-quadrant multiplier with subnanosecond response", IEEE *Journal of Solid-State Circuits*, SC-3:365, 1968

Gilbert, B., "A high performance monolithic multiplier using active feedback", IEEE *Journal of Solid State Circuits*, SC-9 364-373, 1974

Jayant, N.S. and Noll, P., *Digital Coding of waveforms*, Prentice Hall 1984.

Mead, C., A*nalog VLSI and Neural Systems*, Addison-Wesley 1989

Oja, E., "A Simplified Neuron Model as a Principal Component Analyzer", *J. Math. Biol.* Vol 13, pp.267-273, 1982

Rubner, J. and Tavan, P., "A Self-Organizing Network for Principal-Component Analysis", *Europhys. Lett.*, 10(7), pp. 693-698, 1989

Sanger, T., "Optimal unsupervised Learning in a Single-Layer Linear Feedforward Neural Network", *Neural Networks.* Vol 2, pp.459-473, 1989

Tou, J.T. and Gonzales, R.C., *Pattern Recognition Principles*, Addison-Wesley Publ. Comp., 1974

Vittoz, E., "Analog VLSI Implementation of Neural Networks", *Proc. Journées d' électronique*, EPFL, Lausanne, pp.223-250, 1989

Weste, N. and Eshraghian, K. , *Principles of CMOS design*, Addison Wesley, 1985

Williams, R.J., "Feature discovery through error-correction learning", *Technical Report 8501*, San Diego, Inst. of Cognitive Science, University of California UCSD, 1985

Xu, L., Oja, E. and Suen, Ch., "Modified Hebbian Learning for Curve and Surface Fitting", *Neural Networks, Vol.* 5, pp.441-457, 1992