11-1-1993

Accelerated Learning Through a Dynamic Adaptation of the Error Surface

Antonio G. Thome
Purdue University School of Electrical Engineering

Manoel F. Tenorio
Purdue University School of Electrical Engineering

ACCELERATED LEARNING THROUGH A DYNAMIC ADAPTATION OF THE ERROR SURFACE

ANTONIO G. THOME
MANOEL F. TENORIO

TR-EE 93-39
NOVEMBER **1993**

SCHOOL OF ELECTRICAL ENGINEERING
PURDUE UNIVERSITY
WEST LAFAYETTE, INDIANA 47907-1285

Accelerated Learning Through a Dynamic Adaptation of the Error Surface

Antonio G. Thome
and
Manoel F. Tenorio

School of Electrical Engineering
Purdue University
West Lafayette, IN 47907-1285

Accelerated Learning Through a Dynamic Adaptation of the Error Surface

Antonio G. Thome and Manoel F. Tenorio
Parallel Processing Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, IN

Abstract:
In this report we describe a novel technique that accelerates learning processes through a dynamic adaptation of the error surface. The algorithm, here name ARON (Adaptive region of Nonlinearity), implements a generalization of the basic McCulloch-Pitts type of neuron which gives to each unit the ability to automatically adapt its operational region according to the requirements of the problem. The changes on the error surface facilitates the progress of the optimization criterion on its search for a minimum. ARON can be used in addition to and bring benefits to a large class of other optimization schemes.

Key words: Supervised Learning; Learning Accelerating; Neuron model

1. INTRODUCTION

The discovery of efficient learning techniques that allow a network to infer structure directly from data and to build complex representations of the underlying environment, is a major goal of connectionist research. The complexity of the problem imposes severe restrictions on its analysis, understanding, and solution. Because of this, despite the intense effort, the search for new, more reliable, better scaled, and faster algorithms continue to be pursued.

Supervised learning in layered neural networks is posed as a nonlinear optimization problem, where the goal is to minimize an error cost function defined over a set of paired input-output vectors. Optimization techniques are used to modify the connection strengths (weights) so as to represent important features of the task domain. The results of learning can be viewed as a numerical solution to the problem. Iterative methods of optimization consisting of two alternating subcycles is generally used. In the first, from a current point, a search direction is computed, and in the second the objective function is minimized along the chosen direction. This process is repeated until some termination criterion is satisfied.

The Back-Propagation algorithm, introduced by Rumelhart, Hinton, and Williams, [Rum86], among others, is a generalization of the least squares procedure to networks with layers of hidden units between the input and output units. It is based on the generalized delta rule that is itself an extension of the Widrow-Hoff rule for perceptions [Wid60]. The central idea of Back-Propagation is that the derivatives of the cost function with respect to the weights can be computed efficiently by starting with the output layer and propagating backwards, and recursively, through the previous layers. For each input-output pair, a forward-backward cycle is performed. On the forward pass, the activation of each unit is computed, starting at the input units. On the backward pass, starting at the output units, the error measures are computed and backpropagated through the previous layers.

Gradient Descent techniques predominate in connectionism, and steepest descent is by far the most popular method. The algorithm is simple and robust, but is notorious for its slow rate of convergence whenever the objective function is ill conditioned. The complexity of the error surface, as evidenced by flat regions and local minima, imposes severe restrictions on the rate of convergence, stability, and consistency of these algorithms.

2. ACCELERATING TECHNIQUES

The major reasons for the Steepest Descent slow rate of convergence relate to the magnitude of the components of the gradient vector, the direction of the gradient vector, the saturation region, and the step size or learning rate parameter.

The magnitude of the components of the gradient vector may be too small, yielding a minor reduction in the error measure, or too large, yielding overshooting and oscillation problems. The former occurs whenever the error surface is flat along a particular weight dimension so that, the weight adjustment is very small and many steps may be required to achieve a significant reduction in error. The later occurs whenever the error surface is highly curved along a particular dimension so that, the corresponding weight is adjusted by a large amount, possibly overshooting the minimum of the error surface along that weight dimension.

The direction of the gradient in general does not point directly towards the global minimum, which by itself may aggravate the learning problems. From a linear viewpoint it is known that the en-or function is quadratic in the weights, and thus possesses a unique minimum. The orientation and shape of the error surface is determined by auto-correlation matrix R of the input vector, which in this case equals the Hessian, and is derived from the cost function as follows:

$$E(n)= E\{y^*(n) - w^T(n)x(n))^2\}, \tag{1}$$

where

$$E(n)= E\{y^*(n)y^*(n)\} - w^T(n)p - p^Tw(n) + w^T(n)Rw(n), \tag{2}$$

$$p= E\{x(n)y^*(n)\}, \tag{3}$$

$$R= E\{x(n)x^T(n)\}, \tag{4}$$

and

$E\{.\}$ represents the expected value function.

R can be written in terms of its eigenvalues and eigenvectors,

$$R= Q \Lambda Q^{-1}, \tag{5}$$

where

A is a diagonal mamx of eigenvalues

Q is the matrix of corresponding eigenvectors.

The shape of the error surface is defined by the eigenvalues of R. The surface decreases most rapidly in the direction of the eigenvector vector corresponding to the largest eigenvalue (λ_{max}) and most slowly in the direction of the eigenvector corresponding to the smallest eigenvalue

(λ_{min}). The ratio ($\frac{\lambda_{max}}{\lambda_{min}}$) defines a measure of eigenvalue spread, which :rangesfrom one to infinity. When the eigenvalues spread equals one, contours of equal error are circular and the negative of the gradient points toward the minimum. When the spread is larger than one, contours of equal error are elliptical, and the negative of the gradient may not point toward the minimum (fig.1).

An ill-conditioned surface is characterized geometrically by narrow valleys with elliptical contours of high eccentricity, which translates into a wide disparity in the magnitude of the eigenvalues. When nonlinearities are present, the error surface gets more complicated, the Hessian matrix, as observed by Jacobs [Jac88], is no longer constant, and the valleys become curved and may take varying shapes in different regions, thus exacerbating the effects of ill conditioning.

Saturation of the squashing function may create false flat regions and also be a source of ill conditioning. The closer a unit operates to its saturation region, the smaller the weight adjustment. This is because the derivative of the nonlinearity approaches zero, which reduces close to zero the backpropagated error and consequently nullifies the weight adjustment. The derivative of the sigmoid function, for example, is given by

$$o_j (1 - o_j) \tag{6}$$

where o_j is the output activation of the unit j. This derivative goes to zero as the unit's output approaches 0 or 1, resulting in just a tiny fraction of the error being passed back to incoming weights and to units in previous layers. Such a unit may theoretically recover, but it may take a very long time, or never recover due to computational round off and truncation problems. To address this problem, some heuristics are suggested in the literature: to use an error measure that goes to infinity as the derivative goes to zero [Fra87]; to add a constant bias, say .1, to the derivative before using it to scale the error [Fah88], or to employ a rescaling factor to compensate for the reduction in magnitude of the derivative across different layers of the network[Rig91].

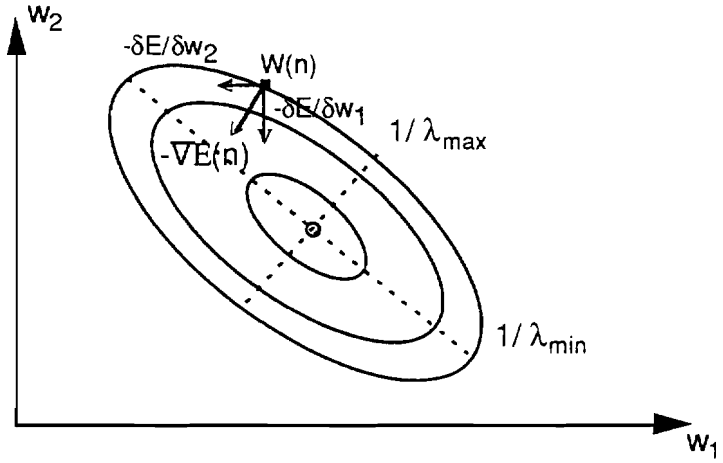

Figure 1 - Error surface contours

The learning rate is commonly taken as a constant; this can also slow the rate of convergence, since a fixed learning rate may not be appropriate for all portions of the error surface. The axis corresponding to the i^{th} eigenvector has length proportional to $1/\lambda_i$, so the component of the

gradient is smaller in the direction of the eigenvector corresponding to λ_{max} (minor axis of the ellipse fig. 1) than in the direction of the eigenvector corresponding to λ_{min} (major axis). Thus, a learning rate that yields moderate steps along the major axis may result in large steps along the minor axis, and oscillations across the minimum for that weight dimension may occur. Vice versa, a learning rate that is appropriate for the minor axis may result in small steps along the major axis, with a correspondingly small reduction in the objective function. In practice this is very likely to happen, since the learning rate is usually set small enough so that overshooting and oscillation are avoided.

Many techniques have been proposed to speed up Back-Propagation: the addition of a momentum term to the generalized delta rule [Rum86]; the adoption of alternative cost functions [Kru91, Sol88], dynamic adaptation of the learning rate[Jac88,Kes58,Sar70]; and rescaling methods [Rig91].

The compensatory rescaling proposed by Rigler is based on the fact that the derivative of the sigmoid imposes a multiplicative reduction factor on the magnitude of the weight adjustments. This factor starts in the range [0, .25] at the output layer and decreases for each successive hidden layer. The compensatory scale factors are constant for each layer, and are based on the assumption that the expected value of the derivative is uniformly distributed in the interval [0,1]. Problems with this approach are that the idea is restricted to sigmoidal-type of squashing functions, the uniform distribution is unlikely to be valid, and values larger than one may frequently occur, reinforcing all gradient magnitudes and, as a consequence, causing more oscillation.

Adaptive learning rates are based on the sign or direction of consecutive changes of a weight, i.e. $\Delta w_i(k)$ and $\Delta w_i(k-1)$. Kesten [Kes58] proposes that if they are opposite in sign, then the weight value is oscillating around its minimum and hence, the learning :rate for that weight should decrease. Saridis [Sar70] extends this notion to both increase (if same sign) and decrease (if of opposite sign) learning rates. Sutton and Barto [Sut81] present variations on this idea. Jacobs [Jac88] introduces two algorithms, the Delta-Delta and the Delta-bar-Delta learning rules, that are minor modifications of the basic delta rule, namely the introduction of an adaptive adjustment to the learning rate.

In Delta-Delta, the weight adjustment is governed by

$$\Delta w(n+1) = -\varepsilon(n) \frac{\partial E}{\partial w(n)}, \tag{7}$$

and

$$\Delta \varepsilon(n) = \gamma \frac{\partial E}{\partial w(n)} \frac{\partial E}{\partial w(n-1)}. \tag{8}$$

This formulation has drawbacks as pointed out by Jacobs himself. Dependling on the value of "γ", the learning rates can become excessively large or small, and even negative. It almost does not help in flat regions (low gradient values), and may increase oscillation in regions with high curvature. To avoid such problems Jacobs suggests the Delta-bar-Delta rule, which has a linear increasing factor, a recursive decreasing factor, and a lower bound to prevent negative values:

$$\Delta\varepsilon(n)=\begin{cases} k & \text{if } \hat{\sigma}(n-1)\sigma(n) > 0 \\ -\alpha\varepsilon(n) & \text{if } \hat{\sigma}(n-1)\sigma(n) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where

$$\sigma(n)=\frac{\partial E}{\partial w(n)}, \tag{10}$$

$$\hat{\sigma}(n)=(1-\theta)\sigma(n)+\theta\hat{\sigma}(n-1). \tag{11}$$

As suggested by Rumelhart at al [Rum86] to speed up convergence without leading to oscillation, ***momentum*** is an extension of the generalized delta rule given by

$$\Delta w(n+1)=-\eta\frac{\partial E}{\partial w(n)}+\mu\Delta w(n). \tag{12}$$

Momentum is considered useful in those cases where the error surface contains long ravines characterized by sharp lateral curvature and an almost flat floor. The sharp curvature tends to cause divergent oscillations along the ravine, which can be mitigated by resmcting the weight adjustments to be very small, but this also slows down the convergence rate. Momentum provides a significant improvement to the standard generalized delta rule since it filters out the high frequency oscillations allowing larger step sizes to be used. However, limitations exist and momentum may generate more problems than it solves, depending on the degree of the curvature and the momentum factor μ.

Equation **12** can be unfolded and rewritten as

$$\Delta w(k+1)=-\eta\sigma_k+\mu^k\Delta w(0)-\eta\sum_{j=1}^{k-1}\mu^j\sigma_{k-j}, \tag{13}$$

Which, by assuming $\mu < 1$ and $\Delta w(0)=0$, reduces to

$$\Delta w(k+1)=-\eta\sigma_k-\eta\sum_{j=1}^{k-1}\mu^j\sigma_{k-j}. \tag{14}$$

For a slightly sloping floor, with almost equal derivatives over several points in a sequence, equation **14** reduces to

$$\Delta w(k+1)=-\eta\frac{1}{1-\mu}\sigma, \tag{15}$$

which shows a lower bound equal to the standard steepest descent learning rate η and no upper bound. On sharp ravines momentum may end up with a sign that is opposite to the sign

of the current derivative, and thus, causing weight adjustment in the wrong direction, as can be seen as follows:

$$\Delta w(k+1) = -\eta\left(\sigma_k + \frac{\mu}{1-\mu}\sigma\right). \qquad (16)$$

3. The Shape of the Error Surface

Least square learning procedures have a simple geometric interpretation. The error surface is constructed over a multi-dimensional space that has an axis for each weight and one extra axis for the error measure. For each combination of weights, the network will have a certain error which can be represented by the height of a point in weight space.

Networks with linear units and no hidden layers always have bowl-shaped error surfaces. The horizontal cross-sections are ellipses and the vertical cross-sections are parabolas. If there is a minimum it is unique and global, and it is guaranteed to be found by any gradient descent method (fig. *2)*. The error surface is in fact the sum of a number of parabolic troughs one for each training pattern. If the output units have a monotonic nonlinearity, then each trough is deformed but no new minima can be created in any trough because the monotonic function cannot change the sign of the gradient . However, when different troughs are added together, it is possible to create local minima. With hidden units, the error surface may contain many local minima, and it is possible for gradient descent to become trapped in any of them. Local minima and slow convergence are the major problems for any gradient descent method (fig. 3).

Within large neural networks, with many nonlinear units, local minima are easily formed and the tour for any learning procedure becomes a very complex task. Successful minimization depends on the starting point as well as on the learning procedure's ability to handle with plateaus, to avoid oscillation in the ravines, and to avoid getting stucked in local minima. The rate of convergence tends to slow down and the desired accuracy is no longer guaranteed.

4. THE ADAPTIVE REGION OF NONLINEARITY SCHEME

ARON (Adaptive Region of Nonlinearity) is novel in the sense that it provides learning speed ups through a dynamic adaptation of the error surface. It is based on a generalization of the basic McCulloch-Pitts type of neuron, it adaptively adjusts the error surface in order to facilitate the progress of a descent algorithm in its search for the global or a good local minimum, and it can applied to many existing learning schemes.

McCulloch-Pitts type of processing units [McC43] are the most widely used node structure in neural networks. It basically consists of a monotonically increasing nonlinear function applied to a linear combination of the inputs (fig 4).

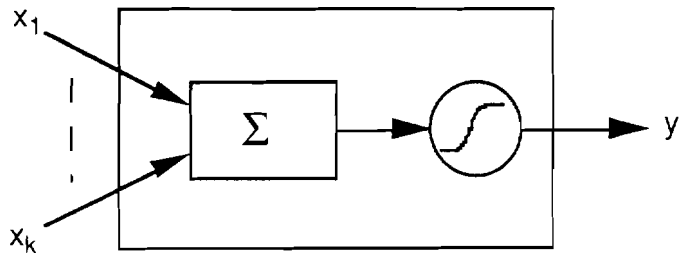

Figure 4 - McCulloch-Pitts Processing Unit Model

McCulloch-Pits units are defined as

$$y = f(\sum_i w_i x_i). \tag{17}$$

A more general model can be constructed by introducing independent gain parameters which may provide a greater degree of freedom and discriminatory power to each node. This model, expressed by

$$y = \beta f(\sum_i \alpha w_i x_i), \tag{18}$$

is schematized in figure 5. The global gains a and β help the node activation to adapt faster and more easily than when every single weight is handled on an individual basis. A compromise among β, a, and the weights must be found automatically by the learning procedure in order to adjust the node to its best role in the network context..and thus reducing the output error.

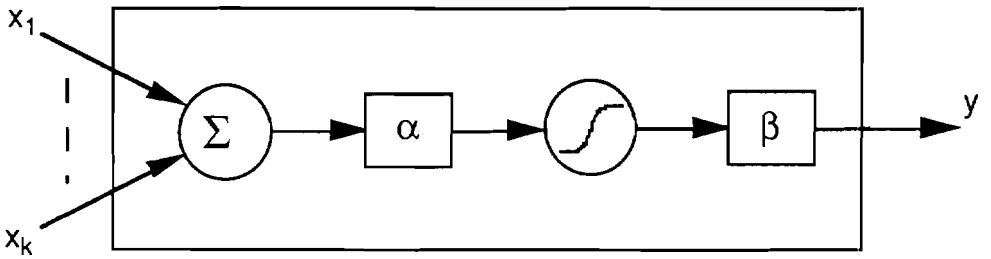

Figure 5 - Processing Unit - General Model

Roughly, it can be claimed that the preceding weights should be able to perform the role of the parameter a, and that the succeeding weights should be able to do the job of β. To an asymptotic behavior this may happen, but for practical purposes and for learning speed up performances it seems intuitive that both parameters (a and β) may provide special help.

Sperduti [Spe92] developed a model in which the parameter β is taken equal to one. He claims that this model not only facilitates fast convergence but also automatically tunes the network structure to the requirements of the problem. In fact, a deals with the slope, or gain ratio, of the nonlinearity (fig 6). Values of a close to zero kill the unit. and very large values turn it into a hard limiter. Sperduti's model is given by

$$y = f(\sum_i \alpha w_i x_i). \tag{19}$$

The model proposed here follows similar reasoning but adopts a more effective strategy. The objective is to adapt the error surface's shape on the fly by allowing each individual node to

adjust its degree of nonlinearity according to the requirements of the problem. The method is based on a minor resmction imposed to the most general formulation in order to get the desired effect:;. It is accomplished by restricting the squashing function to be the hyperbolic tangent and coupling β with α. This results in the following function:

$$y = \frac{1}{\alpha} \tanh\left(\sum_i \alpha w_i x_i\right). \tag{20}$$

This formulation allows each unit to change its degree of nonlinearity, i.e. its region of operation (fig.7), without changing its gain or transfer function characteristic. Small values of α have the property to linearize the node, while large values the property to kill the node. Basically the same two results achieved by Sperduti with the advantage that here the network transfer function characteristics are preserved.

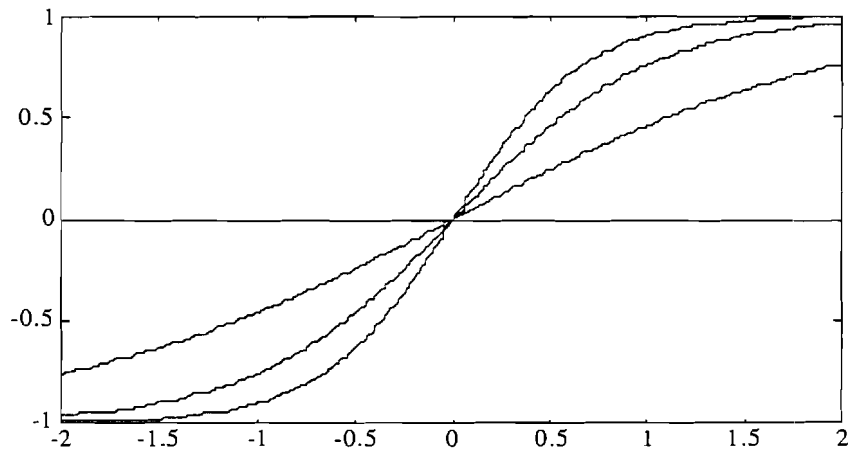

Figure 6 - Sperduti's Nonlinear Function for different values of α: 0.01, .5, 1, and 1.5 from the bottom to the top

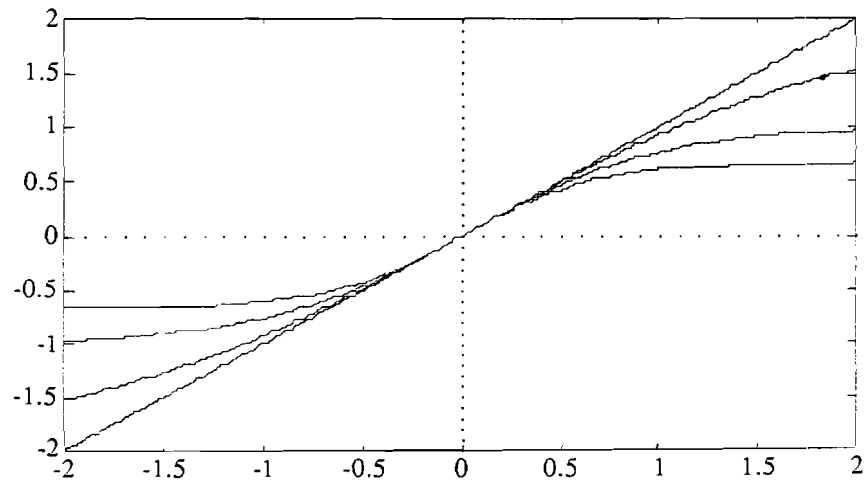

Figure 7 - ARON's Nonlinear Function for different values of a : 1.5, 1, .5, and 0.01 from the bottom to the top

5 - ARON Properties

• The first property provided by this formulation is to dynamically change the shape of the error surface. In fact it can vary from one extreme when all units are linear to the other extreme when all units are highly nonlinear. It will be seen in section 6, network dynamics..that allowing **a** to adapt during the optimization process changes the error surface in a way that seems to accelerate learning and better avoid local minima. However, it is important: to emphasize that the study conducted here focuses exclusively on speed gains and not on asymptotic behavior, which would require deeper analysis and simulations.

• The second major property of ARON relates to the fact that it does not change the original gain, or transfer function, of each node (figs. 7), a property that is not shared by Sperduti's formulation (fig. 6).

• ARON does not penalize the error backpropagation (fig. 8), which remains constant or even increases as new points are introduced into the operational region of the unit. The derivatives with respect to the weights are given by

$$\frac{\partial y_k}{\partial w_{jk}} = x_i \left(1-\left(\tanh\left(\sum_i \alpha w_{ik} x_i\right)\right)^2\right), \tag{21}$$

that is of the same form as the conventional model, in which **a** equals one. However, Sperduti's formulation includes an extra gain (figs. 9) that can act as a reducer when $\alpha<1$ and is passed backward through the entire network as follows:

$$\frac{\partial y_k}{\partial w_{jk}} = \alpha x_j \left(1-\left(\tanh\left(\sum_i \alpha w_{ik} x_i\right)\right)^2\right). \tag{22}$$

• Another ARON property relates to its capability to linearize as well as to remove units, which can be seen as a natural way to automatic tune and minimize the structure of the network. Using a Taylor series expansion it can be shown that the unit behaves linearly for sufficiently small values of **a** and approaches zero for large enough values as follows:

$$y = \frac{1}{\alpha}\left(\sum_i \alpha w_{ik} x_i - \frac{1}{3}\left(\sum_i \alpha w_{ik} x_i\right)^3 + \frac{2}{15}\left(\sum_i \alpha w_{ik} x_i\right)^5 - \frac{17}{315}\left(\sum_i \alpha w_{ik} x_i\right)^7 + \ldots\right) \tag{23}$$

hence

$$\lim_{\alpha \to 0} y = \sum_i w_{ik} x_i, \tag{24}$$

$$\lim_{\alpha \to \infty} y = 0. \tag{25}$$

• Finally, since ARON is only related to the structure of each node, it can be used with any existing learning procedure without restriction. In fact, experiments showed that ARON enhanced the accuracy and speed of all the procedures evaluated.

ARON, by adjusting the operational region of each node, dynamically changes the error surface's shape. Eventually such changes facilitate the progress of the learning procedure and a better local minimum is found. Figure 10 shows the 3-parity problem error surface for different values of a. A reduction in the complexity of the surface is easily observed as the parameter a approaches zero for the output unit case or increases for the hidden units.

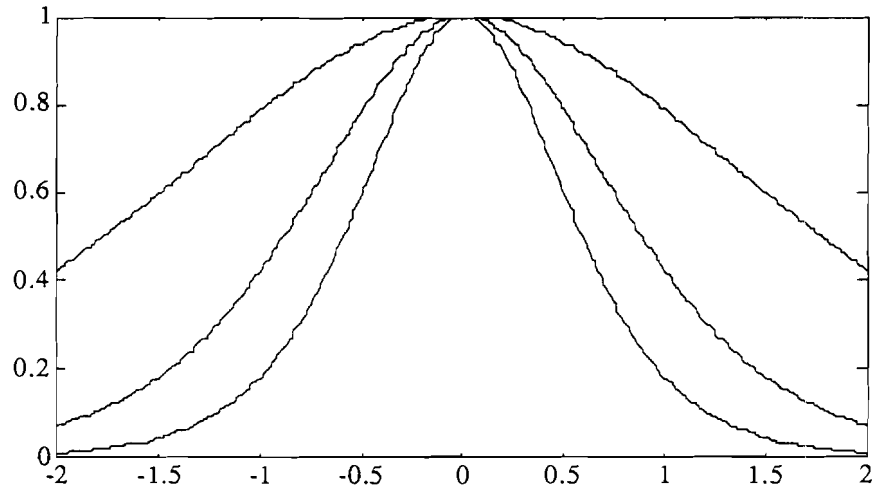

Figure 8 - ARON Function derivative for a = 1.5, 1, and .5 from the bottom to the top

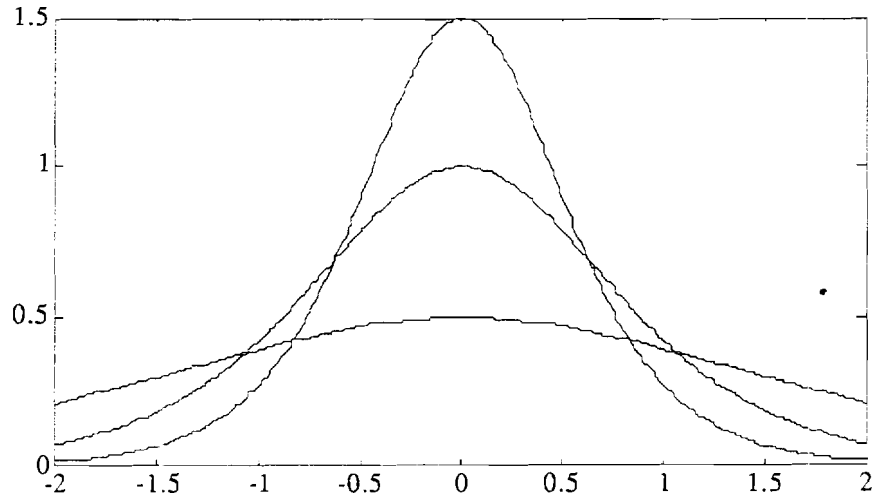

Figure 9 - Sperduti Function derivative for a = 0.5, 1, and 1.5 from the bottom to the top

6 - NETWORK DYNAMICS

The dynamics of the weights remain identical to the conventional model, differing only on the presence of the parameter a on the computation of the output of each unit. With respect to the

dynamics of the parameter α, several different approaches were evaluated. It was found that the best alternative, by far, was to leave it self-adaptive. A gradient descent update formula, as defined in equation 26 and developed in the appendix, was chosen because it provides smoother and smaller changes in the parameter space. The dynamics is then given by

$$\Delta\alpha_i^L = -\frac{\varepsilon}{\alpha_i^L}\left(\delta_i^L S_i^L - \varphi_i^L y_i^L\right), \qquad L = 0 \ldots m \tag{26}$$

where

$$\delta_i^L = \begin{cases} (y_i^* - y_i^o)\, f'(S_i^0); & L = 0 \\ \left(\sum_k \delta_k^{L-1} . w_{ki}^{L-1}\right) f'(S_i^{L-1}); & L = 1 \ldots m \end{cases} \tag{27}$$

$$\varphi_i^L = \begin{cases} (y_i^* - y_i^o); & L = 0 \\ \left(\sum_k \delta_k^{L-1} . w_{ki}^{L-1}\right); & L = 1 \ldots m \end{cases} \tag{28}$$

This formula was applied together with the basic gradient, gradient with momentum, Quickprop [Fah88], and Cubicprop [Cra93] schemes for the evolution of the weights. It was empirically observed that the learning rate for α has to be less than or equal to the learning rate for the weights. The larger the total number of weights or hidden units, the smaller the ratio between the α and w learning rates should be.

Several different initial values for α were med; close to zero, .5, 1, and. random. The best behavior was obtained when all α's started at 1. **A** lower bound of .01 was set in order to prevent negative gains. Figures 11 to 13 show the dynamically evolution of the error surface shape as the learning progresses in a network with a single hidden layer (3 units) and a single unit in. the output layer. Figure 11 shows the accuracy curve for the first 20 epochs, figure 12 shows the evolution of the adaptive parameter α (dotted line for the output unit and other lines for the: hidden units), and figure 13 shows a sequence of changes in the error surface shape.

Each 3-dimensional graph in figure 13 is generated with values of a's and weights (W_c) corresponding to the considered time instant. The current set of weights is linearly combined with two other sets corresponding to two previous solutions of the problem (W_a and W_b). As can be seen in the formulas, the set W_c corresponds to the coordinates (λ=1, β=1). This combination gives:

$$W_x = \lambda W_c + (1-\lambda) W_a \qquad -1.5 \leq \lambda \leq 1.5 \qquad (29)$$

$$W_T = \beta W_x + (1-\beta) W_b \qquad -1.5 \leq \beta \leq 1.5. \qquad (30)$$

A pheriomenon that we call avalanche was found to be responsible for the instability observed on the 9–to–13 time slots. This phenomenon was found characteristic for binary problems, where the desired outputs present a sharp dichotomy [0,1] or [-1,1]. The output unit causes this phenomenon by trying to reduce the error in a faster way through the reduction of its value of a, i.e. linearizing its gain. Suddenly, during an epoch computation, the estimated output for a number of patterns extrapolates the binary thresholds, for example [1, -1], then the gradient vector for the output unit parameter a changes direction, and the unit inverts its tendency reinforcing its nonlinearity. This oscillation is propagated to the previous layers and the entire network oscillates. This avalanche forces a larger adjustment on the weights and the local minimum is eventually avoided. However, the oscillation may last longer or even may not end depending on the ratio between the learning rates for the weights and a.

7. EMPIRICAL RESULTS

As observed by Fahlman [Fah88], there is not yet a widely accepted methodology for measuring and comparing the speed of various connectionist learning algorithms. Some researchers defend their ideas based only on theoretical analysis, and others run few benchmarks to support their claims. Parameter settings and criteria for learning success are other sources of noise in this scenario.

Here the benchmark is realized on the XOR / Parity Problems and for the sake of uniformity, all results are reported in terms of normalized root mean squared error (*Nrmse*). This criterion works well for real problems and seems to be more demanding than other criteria for those binary problems. The *Nrmse* criterion, as stated in equation 31, provides a measure that is independent on the range and on the length of the training set and therefore provides a more convenient basis for comparison. The *Nrmse* is given by

$$\text{Nrmse} = \frac{\sqrt{\frac{(y^*-y)^2}{N}}}{\sigma(y^*)} \qquad (31)$$

where, $\sigma(y^*)$ denotes the standard deviation of y* (desired output). Observe that, by this criterion, if the mean of y* is used as an estimate of y*, i.e y= mean (y*), then the *Nrmse* equals one.

Xor / Parity problems consist of a number of binary inputs, a number of hidden units, and one output unit. The objective is to identify the parity : odd if the number of bits "1" in the input vector is odd, and even otherwise. The experiments reported here consist on the evaluation of the speed up provided by the use of the adaptive parameter a in combination to the pure

gradient, the Quickprop, and the Cubicprop learning algorithms. The notation used is as follows:

G	pure gradient descent procedure
G_α	gradient with ARON parameter (a)
QP	Quickprop basic procedure
QP_α	Quickprop with ARON
Cub	Cubicprop basic procedure
Cub_α	Cubicprop with ARON
avg	average nr. of epochs used to converge to the desired accuracy
std	standard deviation on the number of epochs
median	median of the number of epochs
Hmean	harmonic mean of the number of epochs
No conv	number of times the algorithm did not converge
avg wc	average number of epochs disregarding no convergence cases
best	best run
worst	worst run disregarding no convergence cases
aNrmse	average of the accuracy normalized error
a	ARON gain parameter
μ	momentum factor
ϖ	maximum growth factor
l_α	learning rate for a's
l_w	learning rate for w's (weights)

a) The: XOR problem

A hundred of distinct starting points were used to compute the overall performance of the Gradient, Quickprop, and Cubicprop algorithms alone and with the addition of the ARON parameter (a). As can be seen in table 1, ARON provided a remarkable improvement for the pure gradient procedure. The average number of epochs required to converge to an accuracy of 1e-03 was reduced to less than one third, and one order of magnitude of reduction was achieved in the majority of the runnings. The cubic fitting worked better than the quadratic, and the contribution of a was less significant for the Quickprop algorithm, although it provided an speed up greater than 2 in the majority of the runnings.

b) Three Parity Problem

Similar experiment to the **XOR** was conducted for the 3-Parity case. The algorithms were bounded to perform 500 epochs or to stop whenever the accuracy of 1e-03 was reached, and the statistics were computed over 50 distinct starting points. Three parity with three hidden units showed to a simpler problem than **XOR** with two hidden units.

The statistics in table 2 were computed based on a 3-hidden / 1-output unit network. **A** fixed learning rate of 1 was set for the weights and α (gradient descent procedure), a rate of .5 for both parameters (Quickprop and Cubicprop cases), a maximum growth of 1.75 (Quickprop), and a momentum factor of .9 (Cubicprop). Gradient with α reached the desired accuracy in 86% of the cases, and the best run, with only 18 epochs, was done by Cubicprop. Figure 14 shows the error evolution for one specific starting point, comparing **ARON** against to the basic steepest descent and the Quicprop for the 3-Parity case.

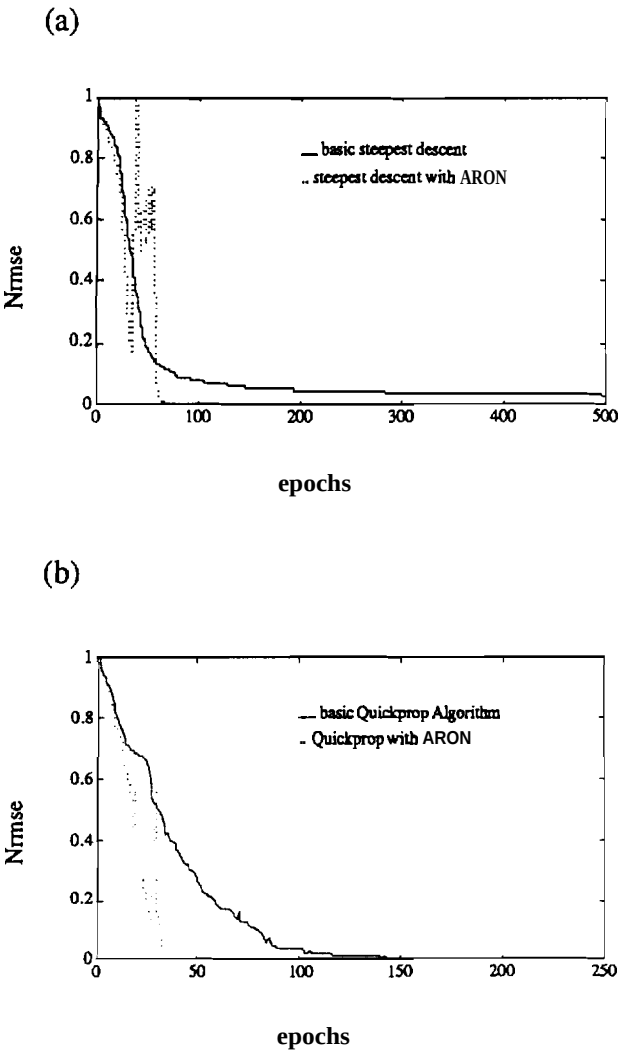

Figure 14 - Steepest Descent and Quickprop Algorithms performance with the addition of the **ARON** parameter on a 3-Parity problem, (a) steepest descent and (b) Quickprop (the instability observed in both graphs is due to the avalanche problem)

Table 1 - **XOR** Problem - statistics over 100 distinct starting points (G - pure gradient descent, G_α - gradient with adaptive a, QP - basic Quickprop, QP_α - Quickprop with adaptive a , Cub - basic Cubicprop, Cub, - Cubicprop with adaptive a) - 2 hidden units, 1 output unit, 500 epochs upper limit

	G	G_α	QP	QP_α	Cub	Cub_α
avg	500	136.46	163.67	151.70	124.85	126.24
std	-	171.60	184.95	202.33	160.84	166.06
median	500	55	67	32	65	55
Hmean	500	64.60	77.15	39.19	50.88	54.98
No conv	100	18	23	24	15	16
avgwc	500	56.65	63.20	41.71	58.64	55.04
test	500	39	40	20	17	20
worst	500	111	93	346	130	249
aNrmse	.1260	.1125	.1425	.1391	.0937	.1002

Table 2 Three Parity Problem - statistics over 50 distinct starting points (G - pure gradient descent, G_α - gradient with adaptive α, QP - basic Quickprop, QP_α - Quickprop with adaptive a, Cub - basic Cubicprop, Cub, - Cubicprop with adaptive a) - 3 hidden units, 1 output unit, 500 epochs upper limit

	G	G_α	QP	QP_α	Cub	Cub_α
avg	500	243.56	63.34	45.68	50.06	49.20
std	-	151.19	21.72	21.76	23.31	18.62
median	500	202	60	38	44	48
Hmean	500	151.04	59.83	38.52	40.87	42.25
No conv	50	7	0	0	0	0
avg wc	500	201.81	63.34	45.68	50.06	49.20
best	500	48	43	21	18	21
worst	500	460	197	111	98	96
aNrmse	.0278	.0011	.0009	.0009	.0009	.0009

Tables 3 through 6 present some analysis on the effect of the learning rate ratio (l_w/l_α), momentum factor ratio (μ/l_α), and the maximum growth ratio (ϖ/l_α). All cases led to the same conclusion in which small values of l_α leads to a more consistent behavior. These results reinforce the avalanche problem described earlier in this chapter and also the intuitive

expectation in which the error surface has to be only smoothly adjusted in order to facilitate the progress of the weights.

Table 3 - Three Parity Problem, the effects of learning rate for the weights and for the parameter a - statistics over 20 distinct starting points for a Gradient descent learning procedure with adaptive parameter a (averaged Nrmse for different combinations of learning rate for the weights and for the parameter a, epoch upper limit of 200)

$l_w \setminus l_\alpha$	.25	.5	.75	1.0	1.5	1.75
.3	8.44e-2	7.84e-2	7.55e-2	7.98e-2	8.51e-2	1.51e-1
.5	2.23e-2	2.43e-2	2.72e-2	5.12e-2	9.96e-2	1.67e-1
1.0	1.33e-3	3.10e-3	1.01e-2	1.11e-3	8.26e-2	1.47e-1
1.5	7.18e-4	1.12e-3	4.25e-2	4.19e-2	4.20e-2	4.21e-2
2.0	1.57e-4	1.70e-4	4.21e-2	4.21e-2	4.14e-2	4.16e-2

Table 4 - Three Parity Problem, the effects of momentum factor and learning rate for the parameter a - statistics over 20 distinct starting points for a Gradient descent learning procedure with adaptive parameter a (averaged Nrmse for different combinations of momentum and for the parameter a, learning rate for the weights set to 1.0, epoch upper limit of 200)

$\mu \setminus l_\alpha$	.25	.5	.75	1.0	1.5	1.75
.3	1.03e-3	9.77e-4	4.20e-2	1.86e-3	8.39e-2	9.23e-2
.5	9.17e-4	9.73e-4	7.55e-4	4.21e-2	1.24e-1	8.31e-2
0.7	1.20e-3	1.28e-3	8.37e-2	8.38e-2	1.34e-1	2.23e-1

Table 5 - Three Parity Problem, the effects of learning rate for the weights and for the parameter a - statistics over 20 distinct starting points for a Quickprop procedure with adaptive parameter a (averaged Nrmse for different combinations of learning rate for the weights and for the parameter a, epoch upper limit of 200)

$l_w \setminus l_\alpha$	.25	.5	1.5	1.75
.3	9.10e-4	6.84e-4	5.53e-4	1.45e-3
.5	7.74e-5	4.51e-3	1.69e-2	1.08e-2
0.7	3.55e-5	7.82e-5	3.32e-2	3.32e-2
1.0	3.31e-2	3.31e-2	4.18e-2	3.79e-2

Table 6 - Three Parity Problem , the effects of maximum growth and learning rate for the parameter a - statistics over 20 distinct starting points for a Quickprop procedure with adaptive parameter a (averaged Nrmse for different combinations of maximum growth and learning rate: for the parameter a , learning rate for the weights set to 1.0, epoch upper limit of 200)

$\varpi \setminus l_\alpha$	.25	.5	1.5	1.75
.3	9.10e-4	6.84e-4	5.53e-4	1.45e-3
.5	7.74e-5	4.51e-3	1.69e-2	1.08e-2
0.7	3.55e-5	7.82e-5	3.32e-2	3.32e-2
1.0	3.31e-2	3.31e-2	4.18e-2	3.79e-2

c) Four Parity Problem

In the four parity problem, the statistics were computed based on a 5-hidden / 1-output unit network. A fixed learning rate of 1 was set for the weights and a (gradient descent procedure), a rate of .5 for both parameters (Quickprop and Cubicprop cases), a maximum growth was set to 1.75 (Quickprop), and a momentum factor of .9 (Cubicprop). Gradient with a performed again much better than the pure gradient procedure reaching the desired accuracy in 62% of the cases, and the best run, with only 75 epochs, was done by Cubicprop.

Table 7 Four Parity Problem - statistics over 50 distinct starting points (G - pure gradient descent, G_α - gradient with adaptive a , QP - basic Quickprop, QP_α - Quickprop with adaptive a , Cub - basic Cubicprop, Cub, - Cubicprop with adaptive a) - 5 hidden units, 1 output unit, 800 epochs upper limit

	G	G_α	QP	QP_α	Cub	Cub_α
avg	g00	572.96	327.46	460.76	443	431
std	-	215.94	261.22	300.87	287.91	325.42
median	800	637	193	291	272	256
Hrnean	800	477.64	211.31	270.45	283.74	201.9
No conv	50	19	10	21	19	21
avg wc	800	433.81	209.32	215.10	224.23	164.97
best	800	171	128	102	86	75
worst	800	746	660	521	389	486
aNrmse	.1275	.0894	.0703	.1221	.1609	.1627

Table 8 - Four Parity Problem, effects of learning rate for the parameter a and for the weights - statistics for a Quickprop procedure with adaptive parameter a (same starting conditions for all runnings, epoch upper limit of 500)

$l_\alpha \backslash l_w$	1.0	1.25	1.5	1.75	2.0
0.00	152	171	146	219	154
0.15	138	242	132	500	137
0.30	193	133	104	500	128
0.50	130	500	500	500	121
0.75	500	500	500	136	212
1.00	500	500	500	500	318
1.50	-	-	500	500	500
2.00	-	-	-	-	500

8 - CONCLUSIONS

ARON is a novel approach that opens a new vein on the search for effective accelerating techniques. The adaptation of the error surface may help the progress of the evolution of the weights and accelerate the learning process. The results obtained are very promising and some possible extensions such as, relaxing the restriction between the gains a and β, and using different learning rates according to layers, may be considered for future analysis.

References

[Cra93] Codringt, C., Thome, A. and Tenorio, M.,"Playing with third oder local approximation", to be submitted to 1994 IEEE NN/Fuzzy conference.

[Fah88] Fahlman, S. E., 1988, "An empirical study of learning speed in back-propagation networks, TR, CMU-CS-88-162.

[Fra87] Franzini, M.A., 1987, "Speech Recognition with backpropagation. Proceedings of Nineth Annual Conference of IEEE Engineering in Medicine and Biology Society.

[Jac88] Jacobs, R. A., 1988, "Increased rates of convergence through learning rate adaptation", Neural Networks, vol 1, pp. 295-307.

[Kes58] Kesten, H., 1958, "Accelerated stochastic approximation", Annals of Mathematical Statistics, 29, pp. 41-59.

[Kru91] Kruschke, J. K., 1991, "Benefits of gain: speed learning and minimal layers in backpropagation learning", IEEE Transactions on Systems, Man, and Cybernetics, vol 21, no. 1, pp. 273-280.

[McC43] McCuloch, W. S. and Pitts, W., 1943, "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysics, vol 5, pp. 115-133.

[Rig91] Rigler, A. K., 1991, "Rescaling of variables in back propagation learning", Neural Networks, vol 4, pp. 225-229.

[Rum86] Rumelhart, D. E., Hinton, G. E., and Williams R. J., 1986, "learning internal representations by error propagation", PDP: Explorations in the Microstructure of Cognition, MIT Press.

[Sar70] Saridis, G. N., 1970, "Learning applied to successive approximation algorithms, IEEE Transactions on Systems, Man, and Cybernetics, vol ssc-6, no. 2, pp. 97-103.

[Sol88] Solla, S. A., 1988, "Accelerated learning in layered neural networks", Complex Systems 2, pp. 625-640.

[Spe92] Sperduti, A. and Starita, A., 1992, " Speed up learning and network optimization with extended back propagation", TR, Dep. of Computer Science, Corso Italia, Pisa, Italy.

[Sut81] Sutton, R. S. and Barto, A. G., 1981, "Toward a modern theory of adaptive networks: Expectation and prediction", Psych. Rev., vol 88, pp. 135-170.

[Wid60] Widrow, B. and Hoff, M. E., 1960, "Adaptive switching circuits", IRE WESCON Conv., record 4, pp. 96-104.

Appendix

ARON dynamics:

$$\Delta\alpha = -\varepsilon \frac{\partial E}{\partial \alpha}$$

where

$$E = \frac{1}{2} \sum_{i=1}^{n} (y_i^* - y_i^o)^2$$

$$y^o = \frac{1}{\alpha^o} \tanh S^o$$

$$S^o = \sum_{i=0}^{h} w_i^o \, y_i^1$$

$$y_i^1 = \frac{1}{\alpha_i^1} \tanh S_i^1$$

$$S_i^1 = \sum_{j=0}^{d} w_{ij}^1 \, X_j$$

a) Output layer (α^o)

$$\frac{\partial E}{\partial \alpha^o} = \frac{\partial E}{\partial y^o} \frac{\partial y^o}{\partial \alpha^o}$$

$$\frac{\partial E}{\partial y^o} = -(y^* - y^o)$$

$$\frac{\partial y^o}{\partial \alpha^o} = \frac{1}{\alpha^o}\left(S^o - S^o \tanh^2(S^o) - y^o\right)$$

hence

$$\Delta(\alpha^o) = \frac{y_i^* - y^o}{\alpha^o}\left((1-\tanh^2(S^o))S^o - y^o\right)$$

b) hidden layer (α^h)

$$\frac{\partial E}{\partial \alpha_i^h} = \frac{\partial E}{\partial y^o}\frac{\partial y^o}{\partial S^o}\frac{\partial S^o}{\partial y_i^1}\frac{\partial y_i^1}{\partial \alpha_i^h}$$

$$\frac{\partial E}{\partial y^o} = -(y^* - y^o)$$

$$\frac{\partial E}{\partial y^o} = \frac{1}{\alpha^o}(1-\tanh^2(S^o))$$

$$\frac{\partial S^o}{\partial y_i^1} = \alpha^o w_i^o$$

$$\frac{\partial y_i^1}{\partial \alpha_i^h} = \frac{1}{\alpha_i^h}\left(S_i^1 - S_i^1 \tanh^2(S_i^1) - y_i^1\right)$$

hence

$$\Delta(\alpha_i^h) = \frac{y_i^* - y^o}{\alpha_i^h}(1-\tanh^2(S^0))\, w_i^o\left((1-\tanh^2(S_i^1))S_i^1 - y_i^1\right)$$

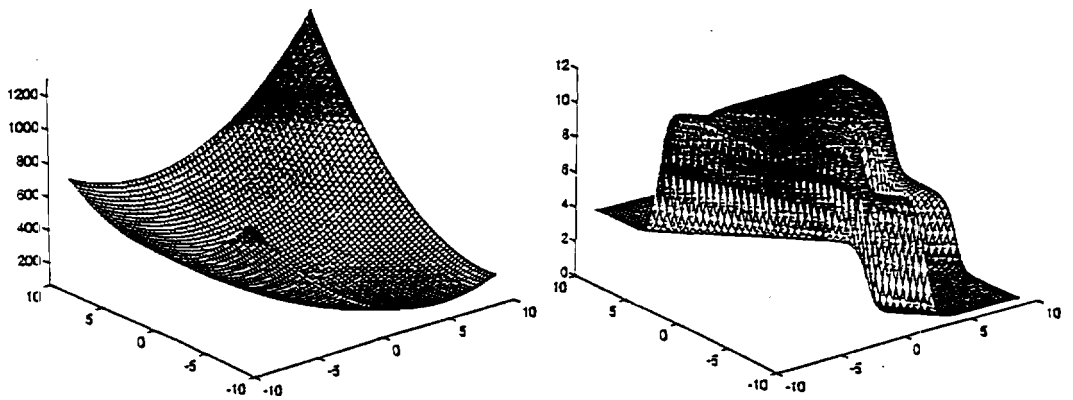

Figure 2 - Error Surface Shape, (a) linear transfer function and (b) nonlinear

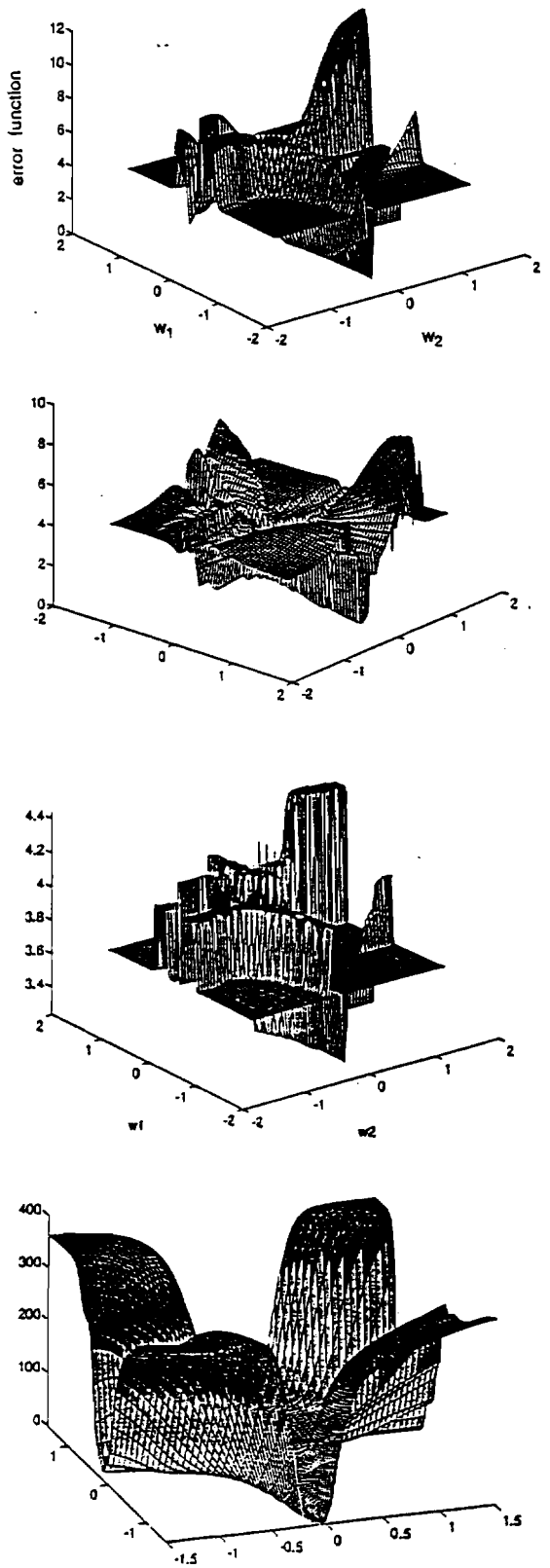

Figure 3 - Error Surface Shape for different values of the parameter a,
(a) α_o= 1, α_h=1; (b) α_o= 1, α_h= 10; (c) α_o=10, α_h=1; and (d) α_o=0.1, α_h=1

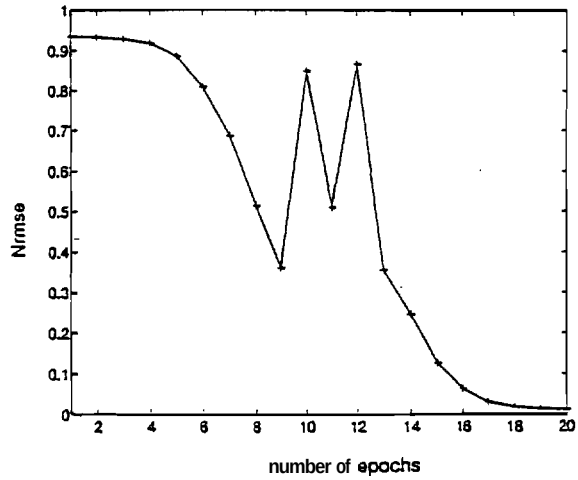

Figure 11 - *Nrmse* evolution for a **3-Parity** problem (3 hidden units, 1 output unit, l_w= 1 and l_a= 1), x axis represents time-step and y axis error value

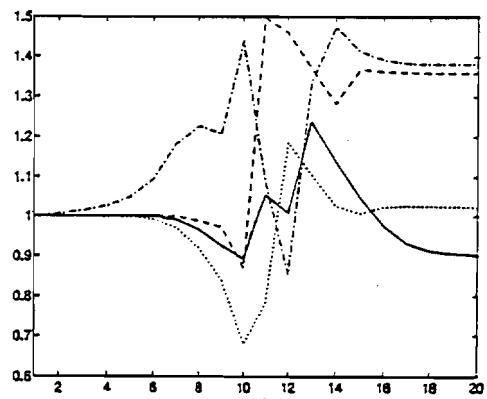

Figure 12 - Evolution of the parameters a (α_o dotted curve, α_h's other curves), x axis represents time-step and y axis represents values of a

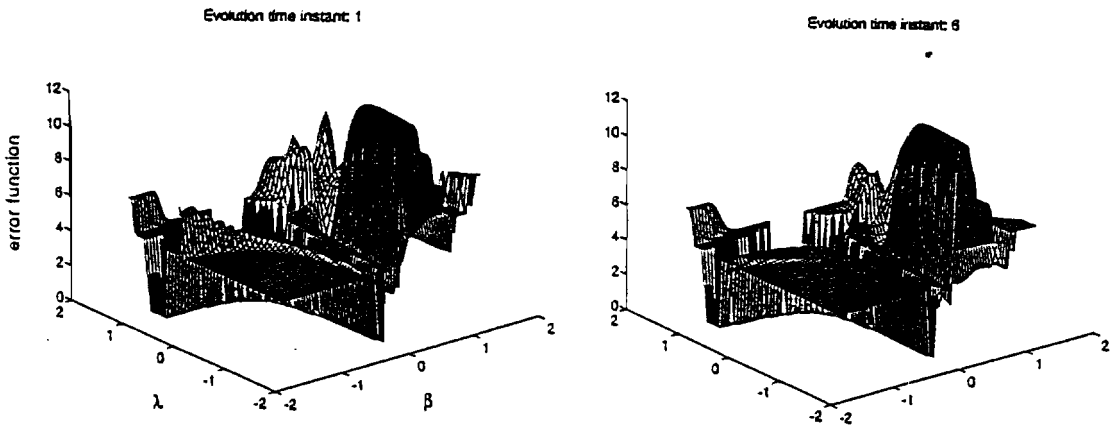

Figure 13 - Error Surface Shape Changing as a function of a (W_c, current set of weights, corresponds to coordinates (1,1), a's values correspond to those in figure 12)

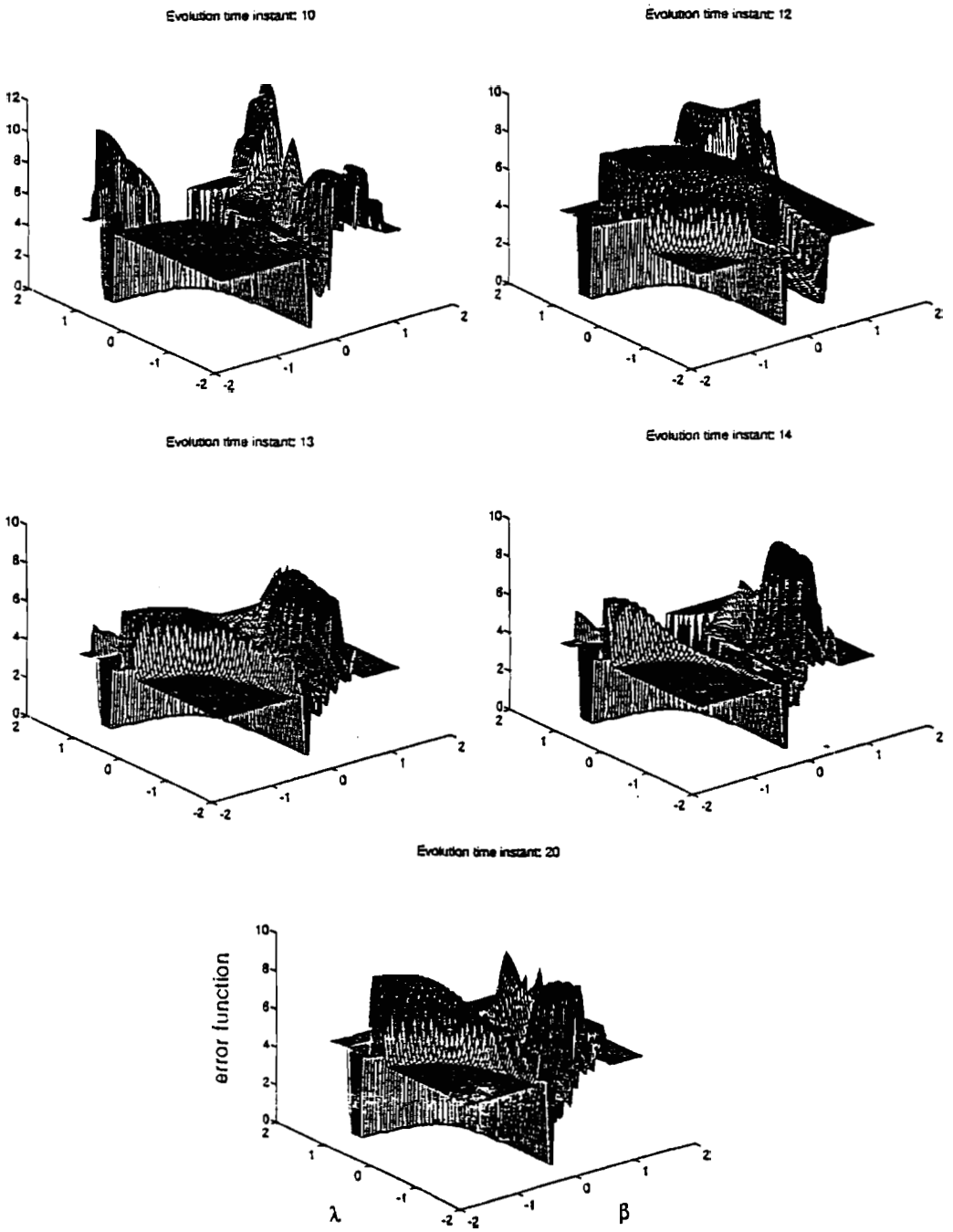

Figure 13 - Continued