Complex Systems 4 (1990) 51–74

Stochastic Approximation and Multilayer Perceptrons: The Gain Backpropagation Algorithm

P.J. Gawthrop
D. Sbarbaro
Department of Mechanical Engineering, The University, Glasgow G12 8QQ, United Kingdom

Abstract. A standard general algorithm, the stochastic approximation algorithm of Albert and Gardner [1], is applied in a new context to compute the weights of a multilayer perceptron network. This leads to a new algorithm, the *gain backpropagation algorithm*, which is related to, but significantly different from, the standard backpropagation algorithm [2]. Some simulation examples show the potential and limitations of the proposed approach and provide comparisons with the conventional backpropagation algorithm.

1. Introduction

As part of a larger research program aimed at crossfertilization of the disciplines of neural networks/parallel distributed processing and control/systems theory, this paper brings together a standard control/systems theory technique (the stochastic approximation algorithm of Albert and Gardner [1]) and a neural networks/parallel distributed processing technique (the multilayer perceptron). From the control/systems theory point of view, this introduces the possibility of adaptive nonlinear transformations with a general structure; from the neural networks/parallel distributed processing point of view, this endows the multilayer perceptron with a more powerful learning algorithm.

The problem is approached from an engineering, rather than a psysiological, perspective. In particular, no claim of physiological relevance is made, nor do we constrain the solution by the conventional loosely coupled parallel distributed processing architecture. The aim is to find a good learning algorithm; the appropriate architecture for implementation is a matter for further investigation.

Multilayer perceptrons are feedforward nets with one or more layers of nodes between the input and output nodes. These additional layers contain hidden units or nodes that are not directly connected to both the input and

output nodes. These hidden units contain nonlinearities and the resultant network is potentially capable of generating mappings not attainable by a purely linear network. The capabilities of this network to solve different kinds of problems have been demonstrated elsewhere [2–4].

However, for each particular problem, the network must be trained; that is, the weights governing the strengths of the connections between units must be varied in order to get the target output corresponding to the input presented. The most popular method for training is *backpropagation* (BP) [2], but this training method requires a large number of iterations before the network generates a satisfactory approximation to the target mapping. Improved rates of convergence arise from the *simulated annealing* technique [5].

As a result of the research carried out by a number of workers during the last two years, several algorithms have been published, each with different characteristics and properties.

Bourland [6] shows that for the autoassociator, the nonlinearities of the hidden units are useless and the optimal parameter values can be derived directly by purely linear technique of *singular value decomposition* (SVD) [7].

Grossman [5] introduced an algorithm called *choice internal representation* for two-layer neural networks, composed of binary linear thresholds. The method performs an efficient search in the space of internal representations; when a correct set is found, the weights can be found by a local perceptron learning rule [8].

Broomhead [9] uses the method of *radial basis function* (RBF) as the technique to adjust the weights of networks with Gaussian or multiquadratic units. The system represents a map from an n-dimensional input space to an m-dimensional output space.

Mitchison [10] studied the bounds on the learning capacity of several networks and then developed the *least action algorithm*. This algorithm is not guaranteed to find every solution, but is much more efficient than BP in the task of learning random vectors; it is a version of the so-called "committee machine" algorithm [11], which is formulated in terms of discrete linear thresholds.

The principal characteristics of these algorithms are summarized in table 1.

The aim of this paper is not to give a physiologically plausible algorithm, but rather to give one that is useful in engineering applications. In particular, the problem is viewed within a standard framework for the optimization of nonlinear systems: the "stochastic approximation" algorithm of Albert and Gardner [1,7].

The method is related to the celebrated BP algorithm [2], and this relationship will be explored in the paper. However, the simulations presented in this paper indicate that our method converges to a satisfactory solution much faster than the BP algorithm [2].

As Minsky and Papert [12] have pointed out, the BP algorithm is a hill-climbing algorithm with all the problems implied by such methods. Our method can also be viewed as a hill-climbing algorithm, but it is more sophis-

Name	Activation function	Type of inputs	Main applications
SVD	linear	continuous or discrete	autoassociators
Choice	binary linear threshold	discrete	logic functions
Least action	linear threshold	discrete	logic functions
RBF	Gaussian or multiquadratic	continuous or discrete	interpolation
BP	sigmoid	discrete or continuous	general

Table 1: Algorithm summary.

ticated than the standard BP method. In particular, unlike the BP method, the cost function approximately minimized by our algorithm is based on past as well as current data.

Perhaps even more importantly, our method provides a bridge between neural network/PDP approaches and well-developed techniques arising from control and systems theory. One consequence of this is that the powerful analytical techniques associated with control and system theory can be brought to bear on neural network/PDP algorithms. Some initial ideas are sketched out in section 5.

The paper is organized as follows. Section 2 provides an outline of the standard stochastic approximation [1,7] in a general setting. Section 3 applies the technique to the multilayer perceptron and considers a number of special cases:

the XOR problem [2],

the parity problem [2],

the symmetry problem [2], and

coordinate transformation [13].

Section 4 provides some simulation evidence for the superior performance of our algorithm. Section 5 outlines possible analytical approaches to convergence. Section 6 concludes the paper.

2. The stochastic approximation algorithm

2.1 Parametric linearization

The stochastic approximation technique deals with the general nonlinear system of the form

$$Y_t = F(\theta, X_t) \tag{2.1}$$

where θ is the parameter vector ($n_p \times 1$) containing the n_p system *parameters*, X_t is the input vector ($n_i \times 1$) containing the n_i system *inputs* at (integer) time t. Y_t is the system output vector at time t. $F(\theta, X_t)$ is a nonlinear, but differentiable, function of the two arguments θ and X_t.

In the context of the layered neural networks discussed in this paper, Y_t is the network output and X_t the input (training pattern) at time t and θ contains the network *weights*. Following Albert and Gardner [1], the first step in the derivation of the learning algorithm is to find a local linearization of equation (2.1) about a nominal parameter vector θ^0.

Expanding F around θ^0 in a first-order Taylor series gives

$$F(\theta^0 + \bar{\theta}, X_t) = F(\theta^0, X_t) + F'(\theta^0, X_t)\bar{\theta} + \epsilon \tag{2.2}$$

where

$$F' = \frac{\partial F}{\partial \theta} \tag{2.3}$$

and ϵ is the approximation error representing the higher terms in the series expansion.

Defining

$$\tilde{Y}_t = F(\theta^0 + \bar{\theta}, X_t) - F(\theta^0, X_t) \tag{2.4}$$

and

$$\tilde{X}_t = F'(\theta^0, X_t)^T \tag{2.5}$$

equation 2.2 then becomes

$$\tilde{Y}_t = \tilde{X}_t^T \theta + \epsilon \tag{2.6}$$

This equation forms the desired linear approximation of the function $F(\theta, X_t)$, with ϵ representing the approximation error, and forms the basis of a least-squares type algorithm to estimate θ.

2.2 Least-squares and the pseudoinverse

This section applies the standard (nonrecursive) and recursive least-square algorithm to the estimation of θ in the linearized equation (2.6), but with one difference: a *pseudoinverse* is used to avoid difficulties with a non-unique optimal estimate for the parameters, which manifests itself as a singular data-dependent matrix [7].

The standard least-square cost function with exponential discounting of data is

$$V_T(\theta) = \frac{1}{T}\sum_{t=1}^{T} \lambda^{T-t}[\tilde{Y}_t - \tilde{X}_t^T\theta]^2 \tag{2.7}$$

where the *exponential forgetting factor* λ gives different weights to different observations and T is the number of presentations of the different training sets.

Using standard manipulations, the value of the parameter vector $\hat{\theta}_T$ minimizing the cost function is obtained from the linear algebraic equation

$$S_T \hat{\theta}_T = \sum_{t=1}^{T} \lambda^{T-t} \tilde{X}_t \tilde{Y}_t \tag{2.8}$$

where the matrix $S_T(n_p \times n_p)$ is given by

$$S_T = \sum_{t=1}^{T} \lambda^{T-t} \tilde{X}_t \tilde{X}_t^T \tag{2.9}$$

When applied to the layered neural networks discussed in this paper, S_T will usually be singular (or at least nearly singular), thus the estimate will be non-unique. This non-uniqueness is of no consequence when computing the network output, but it is vital to take it into account in the computation of the estimates.

Here the minimum norm solution for $\hat{\theta}_T$ is chosen as

$$\hat{\theta}_T = S_T^+ \sum_{t=1}^{T} \lambda^{T-t+1} \tilde{X}_t \tilde{Y}_t \tag{2.10}$$

where S_T^+ is a pseudoinverse of S_T.

In practice, a *recursive* form of (2.10) is required. Using standard manipulations

$$\hat{\theta}_{t+1} = \hat{\theta}_t + S_t^+ \tilde{X}_t e_t \tag{2.11}$$

where the *error* e_t is given by

$$e_t = (\tilde{Y}_t - \hat{\theta}_t^T \tilde{X}_t) \tag{2.12}$$

but, using (2.4) and (2.5),

$$\tilde{Y}_t - \hat{\theta}_t^T \tilde{X}_t \simeq Y_t - F(\hat{\theta}_t, X_t) \tag{2.13}$$

and finally (2.11) becomes

$$\hat{\theta}_{t+1} = \hat{\theta}_t + S_t^+ \tilde{X}_t (Y_t - F(\hat{\theta}, X_t)) \tag{2.14}$$

S_t can be recursively updated as

$$S_t = \lambda S_{t-1} + \tilde{X}_t \tilde{X}_t^T \tag{2.15}$$

Indeed, S_t^+ itself can be updated directly [7], but the details are not pursued further here.

Data is discarded at an exponential rate with time constant τ given by

$$\tau = \frac{1}{1-\lambda} \tag{2.16}$$

where τ is in units of samples. This feature is necessary to discount old information corresponding to estimates far from the convergence point and thus inappropriate to the linearized model about the desired nominal parameter vector θ^0.

3. The multilayer perceptron

At a given time t, the ith layer of a multilayer perceptron can be represented by

$$y_i = W_i^T x_i \tag{3.1}$$

where x_i is given by

$$x_i = \begin{pmatrix} \breve{x}_i \\ 1 \end{pmatrix} \tag{3.2}$$

and $\breve{x}_i$ by

$$\breve{x}_i = \bar{f}(y_{i-1}) \tag{3.3}$$

The unit last element of x_i corresponds to an offset term when multiplied by the appropriate weight vector. Equation (3.1) describes the *linear* part of the layer where x_i $(n_i+1 \times 1)$ contain the n_i inputs to the layer together with 1 in the last element, W_i $(n_i+1 \times n_{i+1})$ maps the input to the linear output of the layer, and $\breve{W}_i$ $(n_i \times n_{i+1})$ does not contain the offset weights. Equation (3.3) describes the *nonlinear* part of the layer where the function $\bar{f}$ maps each element of y_{i-1}, the linear output of the previous layer, to each element of $\breve{x}_i$. A special case of the nonlinear transformation of equation (3.3) occurs when each element x of the vector $\breve{x}_i$ is obtained from the corresponding element y of the matrix y_{i-1} by the same nonlinear function:

$$x = f(y) \tag{3.4}$$

There are many possible choices for $f(y)$ in equation (3.4), but, like the backpropagation algorithm [2], our method requires $f(y)$ to be differentiable. Typical functions are the weighted *sigmoid* function [2]

$$f(y) = \frac{1}{1 + e^{-\alpha y}} \tag{3.5}$$

and the weighted *hyperbolic tangent* function

$$f(y) = \tanh(\alpha y) = \frac{e^{\alpha y} - e^{-\alpha y}}{e^{\alpha y} + e^{-\alpha y}} \tag{3.6}$$

The former is appropriate to logic levels of 0 and 1; the latter is appropriate to logic levels of -1 and 1. The derivative of the function in equation (3.5) is

$$f'(y) = \frac{dx}{dy} = \alpha x(1 - x) \tag{3.7}$$

and that of the function in equation (3.6) is

$$f'(y) = \frac{dx}{dy} = \alpha(1 + x)(1 - x) = \alpha(1 - x^2) \tag{3.8}$$

The multilayer perceptron may thus be regarded as a special case of the function displayed in equation (2.1) where, at time t, $Y_t = x_N$, θ^0 is a column vector containing the elements of all the W_i matrices and $X_t = x_1$. To apply the algorithm in section 2, however, it is necessary to find an expression for $\tilde{X}_t = F'(\theta^0, X_t)$ as in equation (2.6). Because of the simple recursive feedforward structure of the multilayer perceptron, it is possible to obtain a simple recursive algorithm for the computation of the elements of $\tilde{X}_t$. The algorithm is, not surprisingly, related to the BP algorithm.

As a first step, define the *incremental gain* matrix G_i relating the ith layer to the net output evaluated for a given set of weights and net inputs.

$$G_i = \frac{\partial x_N}{\partial \breve{x}_i} \tag{3.9}$$

Using the chain rule [14], it follows that

$$G_i = G_{i+1} \frac{\partial \breve{x}_{i+1}}{\partial y_i} \frac{\partial y_i}{\partial \breve{x}_i} \tag{3.10}$$

and using equations (3.1) and (3.3)

$$G_i = G_{i+1} \bar{f}'(y_i) \breve{W}_i^T \tag{3.11}$$

Equation (3.11) will be called the *gain backpropagation algorithm* or GBP. By definition,

$$G_N = \frac{\partial \breve{x}_N}{\partial \breve{x}_N} = I_{n_i} \tag{3.12}$$

where I_{n_i} is the $n_i \times n_i$ unit matrix.

Applying the chain rule once more,

$$\frac{\partial x_N}{\partial W_i} = \frac{\partial x_N}{\partial \breve{x}_{i+1}} \frac{\partial \breve{x}_{i+1}}{\partial W_i} \tag{3.13}$$

Substituting from equations (3.1), (3.3), and (3.9), equation (3.13) becomes

$$\frac{\partial x_N}{\partial W_i} = G_{i+1} \frac{\partial \breve{x}_{i+1}}{\partial W_i} \tag{3.14}$$

3.1 The algorithm

Initially:

1. Set the elements of the weight matrices W_i to small random values and load the corresponding elements into the column vector θ_0.
2. Set λ between .95 and .99 and the matrix S_0 to $s_0 I$ where I is a unit matrix of appropriate dimension and s_0 a small number.

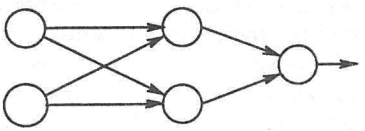

Figure 1: Architecture to solve the XOR problem.

At each time step:

1. Present an input $X_t = x_1$.
2. Propagate the input *forward* through the network using equations (3.1) and (3.3) to give x_i and y_i for each layer.
3. Given x_i for each layer, propagate the incremental gain matrix G_i *backward* using the gain backpropagation of equations (3.11) and (3.12).
4. Compute the partial derivative $\partial x_N / \partial W_i$ using equation (3.14) and load the corresponding elements into the column vector $\tilde{X}$.
5. Update the matrix S_t using (2.15) and find its pseudoinverse S_t^+ (or update S_t^+ directly [7]).
6. Compute the output error e_t from equation (2.12).
7. Update the parameter vector $\hat{\theta}_t$ using equation (2.14).
8. Reconstruct the weight matrices W_i from the parameter vector $\hat{\theta}_t$.

3.2 The XOR problem

The architecture for solving the XOR problem with two hidden units and no direct connections from inputs to output is shown in figure 1.

In this case, the weight and input matrices for layer 1 are of the form

$$W_1 = \begin{pmatrix} w_{111} & w_{112} \\ w_{121} & w_{122} \\ w_{131} & w_{132} \end{pmatrix}, \quad x_1 = \begin{pmatrix} x_{11} \\ x_{12} \\ 1 \end{pmatrix} \tag{3.15}$$

The last row of the matrix in equation (3.15) corresponds to the offset terms. The weight and input matrices for layer 2 are of the form

$$W_2 = \begin{pmatrix} w_{211} \\ w_{221} \\ w_{231} \end{pmatrix}, \quad x_2 = \begin{pmatrix} x_{21} \\ x_{22} \\ 1 \end{pmatrix} \tag{3.16}$$

Once again, the last row of the matrix in equation (3.16) corresponds to the offset term.

Applying the GBP algorithm (3.11) gives

$$G_2 = \breve{W}_2^T \bar{f}'(y_2) = \begin{pmatrix} w_{211} \\ w_{221} \end{pmatrix}^T \bar{f}'(y_2) \tag{3.17}$$

Hence, using (3.14) and (3.12),

$$\frac{\partial x_3}{\partial W_2} = x_2 \bar{f}'(y_2) = x_2 x_3 (1 - x_3) \tag{3.18}$$

and using (3.14) and (3.17)

$$\frac{\partial x_3}{\partial W_1} = G_2 \frac{\partial \breve{x}_2}{\partial W_1} \tag{3.19}$$

Thus, in this case, the terms $\tilde{X}$ and θ linearized equation (2.6) are given by

$$\tilde{X} = \begin{pmatrix} x_{11}x_{21}(1-x_{21})x_3(1-x_3)w_{211} \\ x_{12}x_{21}(1-x_{21})x_3(1-x_3)w_{211} \\ x_{21}(1-x_{21})x_3(1-x_3)w_{211} \\ x_{11}x_{22}(1-x_{22})x_3(1-x_3)w_{221} \\ x_{12}x_{22}(1-x_{22})x_3(1-x_3)w_{221} \\ x_{22}(1-x_{22})x_3(1-x_3)w_{221} \\ x_{21}x_3(1-x_3) \\ x_{21}x_3(1-x_3) \\ x_3(1-x_3) \end{pmatrix}, \quad \theta = \begin{pmatrix} w_{111} \\ w_{121} \\ w_{131} \\ w_{112} \\ w_{122} \\ w_{132} \\ w_{211} \\ w_{221} \\ w_{231} \end{pmatrix} \tag{3.20}$$

3.3 The parity problem

In this problem [2], the outputs of the network must be TRUE if the input pattern contains an odd number of 1s and FALSE otherwise. The structure is shown in figure 5; there are 16 pattern and 4 hidden units.

This is a very demanding problem, because the solution takes full advantage of the nonlinear characteristics of the hidden units in order to produce an output sensitive to a change in only one input.

The linearization is similar to that of the XOR network except that W_1 is 4×4, W_2 is 4×1, and input vector x_1 is 4×1.

3.4 The symmetry problem

The output of the network must be TRUE if the input pattern is symmetric about its center and FALSE otherwise. The appropriate structure is shown in figure 10.

The linearization is similar to that of the XOR network except that W_1 is 6×2, W_2 is 2×1, and input vector x_1 is 6×1.

Problem	Gain (BP)	Momentum (BP)	λ (BP)
XOR	0.5	0.9	0.95
Parity	0.5	0.9	0.98
Symmetry	0.1	0.9	0.99
C. transform	0.02	0.95	0.99

Table 2: Parameters used in the simulations.

x_1	x_2	y
0	0	0.9
1	0	0.1
0	1	0.1
1	1	0.9

Table 3: Patterns for the XOR problem.

3.5 Coordinate transformation

The Cartesian endpoint position of a rigid two-link manipulator (figure 14) is given by

$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) \tag{3.21}$$

$$y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) \tag{3.22}$$

where $[x, y]$ is the position in the plane, $L_1 = 0.3m$, $L_2 = 0.2m$ are the lengths of the links. The joints angles are θ_1, θ_2.

Equation (3.21) and (3.22) show that it is possible to produce this kind of transformation using the structure shown in figure 15.

The linearization is similar to that of the XOR network except that W_1 is 2×10, W_2 is 10×1, and input vector x_1 is 10×1.

4. Simulations

The simple multilayer perceptrons discussed in section 3 were implemented in Matlab [15] and a number of simulation experiments were performed to evaluate the proposed algorithm and compare its performance with the conventional BP algorithm. The values of the parameters used in each problem are shown in table 2.

4.1 The XOR problem

The training patterns used are shown in table 3. The conventional BP and GBP algorithms were simulated. Figure 2 shows the performance of the BP algorithm, and figure 3 shows the performance of the GBP algorithm.

The reduction in the number of iterations is four-fold for the same error.

The effect of discounting past data is shown in figure 4; with a forgetting factor equal to 1, the algorithm does not converge.

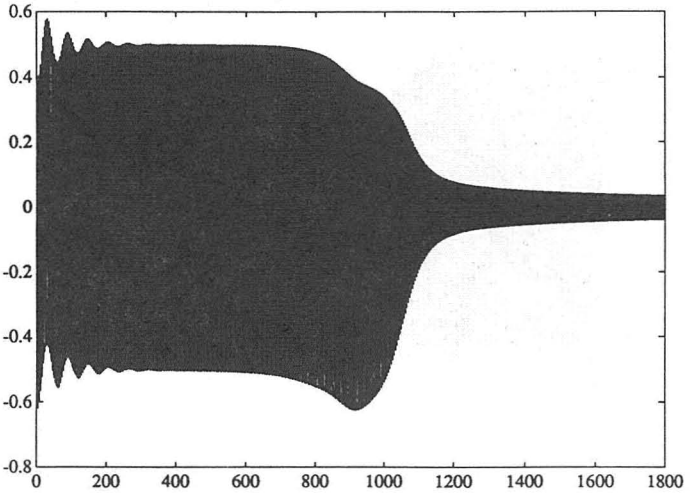

Figure 2: Error for the BP algorithm.

4.2 The parity problem

The output target value was set to 0.6 for TRUE and 0.4 for FALSE. The algorithm used to calculate the pseudoinverse is based on the singular value decomposition [15]; with this formulation it is necessary to specify a lower bound for the matrix eigenvalues. In this case the value was set to 0.001. With the presentation of 10 patterns, the algorithm converged in 70 presentations of each pattern (see figure 6). Figure 8 shows the error ϵ and figure 9 shows the trace of S^+ for each iteration when 16 patterns are presented in a random sequence. The GPA algorithm converged at about 80 presentations of each pattern, compared with about 2,400 for the BP algorithm (figure 7). Note that the solution reached by the GBP algorithm is different than that of the BP algorithm. The difference between the results obtained with both methods could be explained on the bases of 1) the different values assigned to the expressions TRUE and FALSE and 2) a difference in the initial conditions.

4.3 The symmetry problem

In this case there are 64 training patterns. The forgetting factor was set $\lambda = 0.99$, higher than that used in the parity problem because in this case there were more patterns. The targets were set at 0.7 for TRUE and 0.3

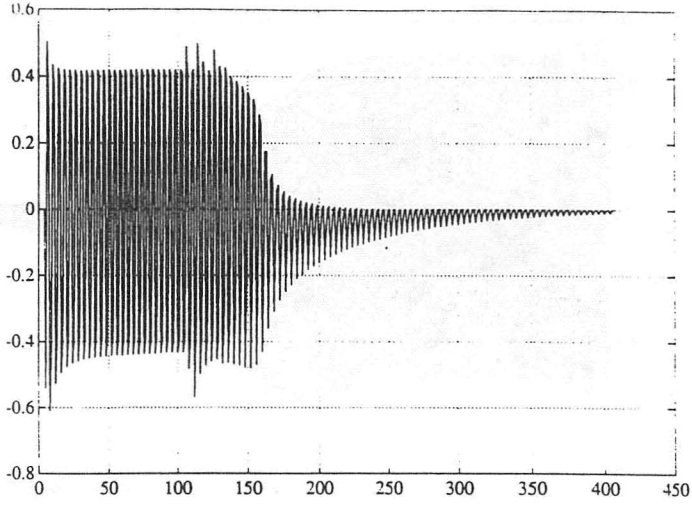

Figure 3: Error for the GBP algorithm.

for FALSE. Errors are shown in figures 11 and 12. It can be seen that the rate of convergence is very fast compared to that of the BP algorithm. The convergence depends on initial conditions and parameters, including the forgetting factor λ and the output values assigned to the targets. The solution found by the GBP algorithm is similar to that found by the BP algorithm.

4.4 Coordinate transformation

The neural network was trained to do the coordinate transformation in the following range of $\theta_1 = [0, 2\pi]$ and $\theta_2 = [0, \pi]$; from this region a grid of 33 training values were used. The evolution of the square error (measured in m) using BP and GBP is shown in figures 16 and 17. The GBP algorithm converges faster than the BP algorithm and reaches a lower minimum. Table 4 shows the number of presentations required for both algorithms to reach a sum square error of 0.02. Figures 18 and 19 show the initial and final transformation surface reached by the network.

4.5 Summary of simulation results

The results for the four sets of experiments appear in table 4.

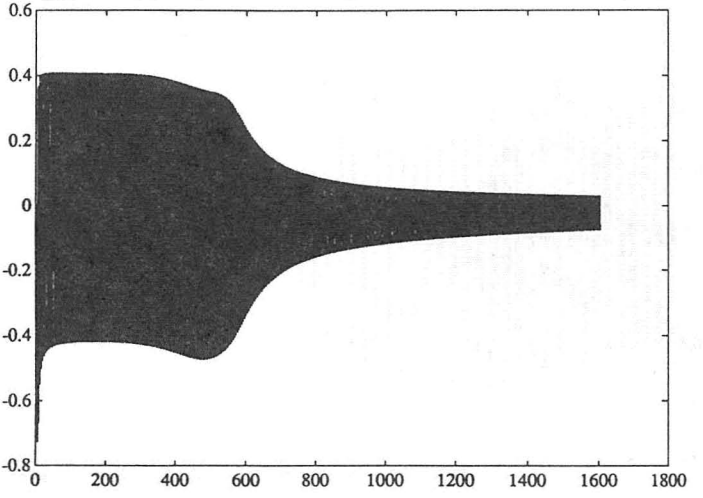

Figure 4: Error for the GBP algorithm with forgetting factor = 1.

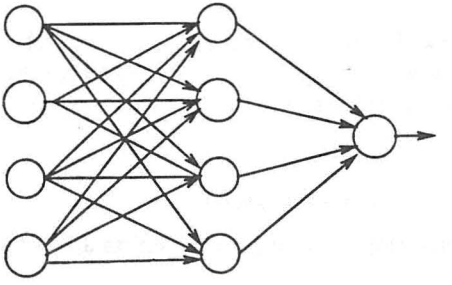

Figure 5: Architecture to solve the parity problem.

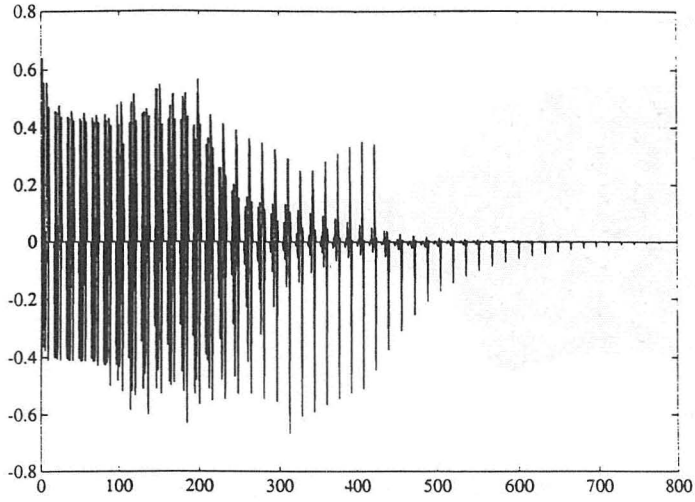

Figure 6: Error for the GBP algorithm (presentation of only ten patterns).

Problem	N(BP)	N(GBP)
XOR	300	70
Parity	2,425	81
Symmetry	1,208	198
C. Transform	3,450	24

Table 4: Simulation results, number of presentation.

The most difficult problem to solve with the GBP algorithm was the parity problem, in which case it was necessary to run several simulations to get convergence. The other problems were less sensitive to the parameters of the algorithm.

5. Comments about convergence

Following the derivations of Ljung [16], if exists a θ^0 such that

$$e_t = Y_t - F(\theta^0, X_t) = \epsilon_t \tag{5.1}$$

where ϵ_j is a white noise, and introducing

$$\tilde{\theta}_t = \hat{\theta}_t - \theta^0 \tag{5.2}$$

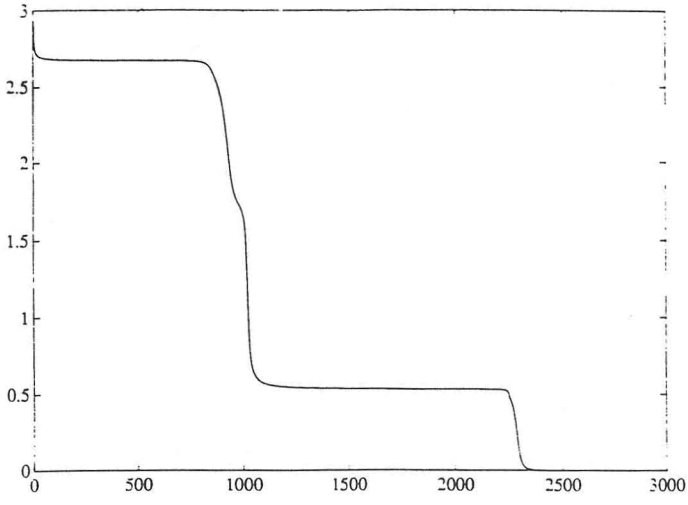

Figure 7: Square error for the BP algorithm.

in equation (2.11),

$$S_t\tilde{\theta}_{t+1} = S_t\tilde{\theta}_t + \tilde{X}_t e_t \tag{5.3}$$
$$= \lambda S_t\tilde{\theta}_t + \tilde{X}_t\tilde{X}_t^T\tilde{\theta}_t + \tilde{X}_t e_t \tag{5.4}$$

Using the expression

$$S_t = \sum_j \beta(j,t)\tilde{X}_j\tilde{X}_j^T \tag{5.5}$$

and setting

$$\lambda^{t-j} = \beta(j,t) \tag{5.6}$$
$$S_t\tilde{\theta}_{t+1} = \beta(0,t)S_0\tilde{\theta}_0 + \sum_j \beta(j,t)\tilde{X}_j[\tilde{X}_j^T\tilde{\theta}_j + e_j] \tag{5.7}$$

and in general

$$e_t = Y_t - F(\theta_t, X_t) \tag{5.8}$$

considering a linear approximation around θ^0

$$e_t \simeq \epsilon_t - \tilde{X}_t^T\tilde{\theta}_t \tag{5.9}$$

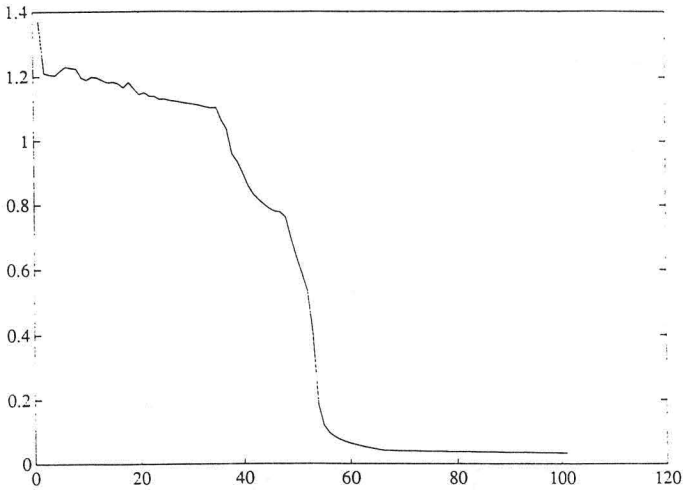

Figure 8: Square error for the GBP algorithm.

So, equation (5.7) becomes

$$S_t\tilde{\theta}_{t+1} = \sum_j \beta(j,t)\tilde{X}_j\epsilon_j \tag{5.10}$$

$$\tilde{\theta}_{t+1} = S_t^+ \sum_j \beta(j,t)\tilde{X}_j\epsilon_j \tag{5.11}$$

Then $\tilde{\theta}_t \to 0$ as $t \to \infty$, representing a global minimum if

1. S_t has at least one nonzero eigenvalue.
2. $\sum_{k=1}^{\infty} \|\tilde{X}_k\| = \infty$ and some of the following conditions are met.
3. The input sequence X_t is such that $\tilde{X}_t$ is independent of ϵ_t and
4. (a) ϵ_t is a white noise and is independent on what has happened up to the time $n-1$ or

   (b) $\epsilon_t = 0$.

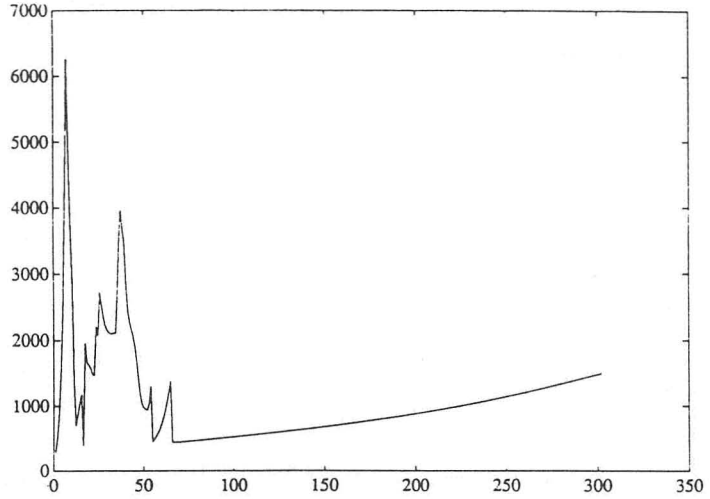

Figure 9: Trace of the covariance matrix for the GBP algorithm.

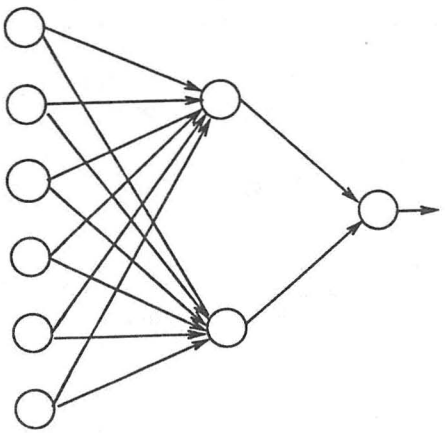

Figure 10: Architecture to solve the symmetry problem.

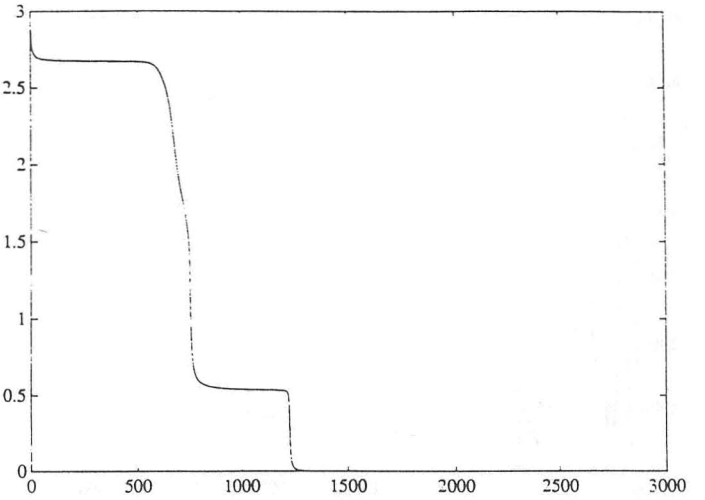

Figure 11: Square error for the BP algorithm.

Comments:

1. This analysis is based in two assumptions: first that $\tilde{\theta}_0$ is close to θ^0, that is, the analysis is only local, and second that an architecture of the NN that can approximate the behavior of the training set. Even though these assumptions seem quite restrictive, it is possible to have some insight over the convergence problem.

2. The conditions (1) and (2) are the most important from the solution point of view, because if they are not met it is possible to have a local minimum.

3. The condition (2) means that the derivatives must be different from 0, that is, all the outputs of the units cannot be 1 or 0 at the same time. That means if the input is bounded will be necessary bound the parameters; for example, one way to accomplish that is to change the interpretation of the output [2].

4. If S is not invertible, the criterion will not have a unique solution. It will have a "valley" and certain linear combination of the parameters will not converge or converge an order of magnitude more slowly [17]. There are many causes for the nonexistence of S inverse, such as the model set contains "too many parameters" or when experimental conditions

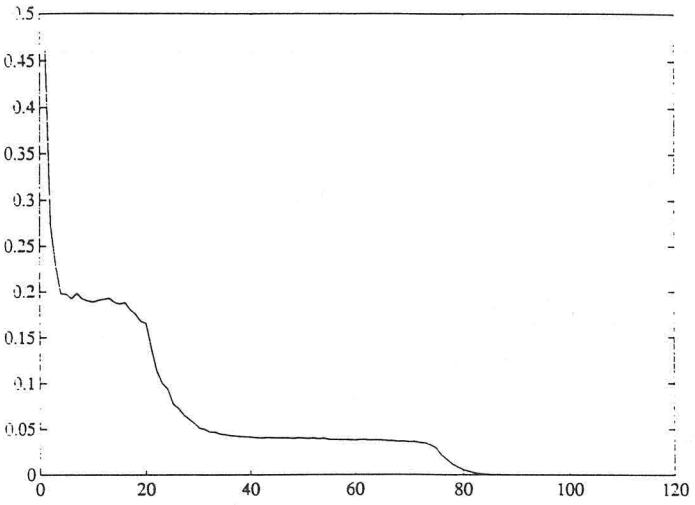

Figure 12: Square error for the GBP algorithm.

are such that individual parameters do not affect the predictions [17]. These effects arise during the learning process, because only some units are active in some regions contributing to the output.

5. The condition (4b) arise when the structure neural network's structure can match the structure of the system.

6. Condition (3) shows that the order of presentation can be important to get convergence of the algorithm.

6. Conclusions

The stochastic approximation algorithm presented in this paper can successfully adjust the weights of a multilayer perceptron using many less presentations of the training data than required for the BP method. However, there is an increased complexity in the calculation.

A formal proof of the convergence properties of the algorithm in this context has not been given; instead, some insights into the convergence problem, together with links back to appropriate control and systems theory, are given. Moreover, our experience indicates that the GPA algorithm is better than the BP method at performing continuous transformations. This difference is produced because in the continuous mapping the transformation is smooth

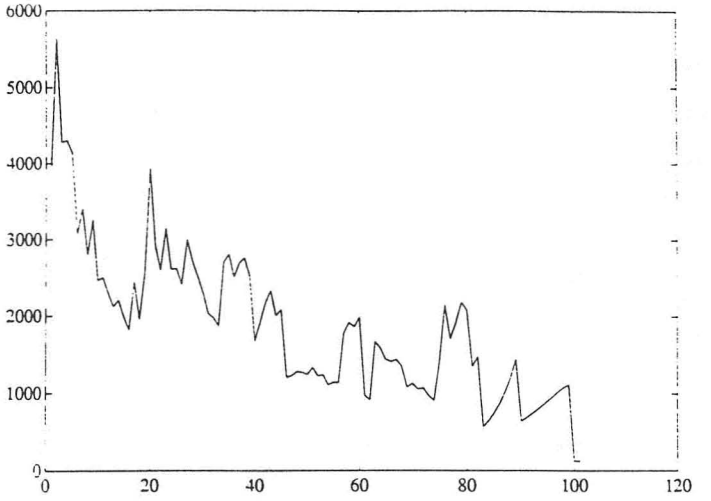

Figure 13: Trace of the covariance matrix for the GBP algorithm.

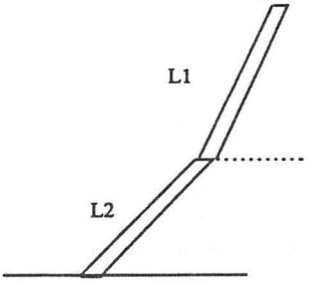

Figure 14: Cartesian manipulator.

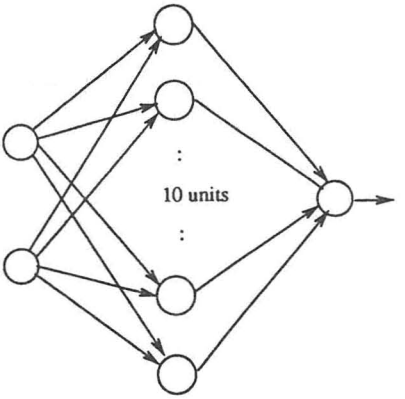

Figure 15: Architecture to solve the coordinate transformation problem.

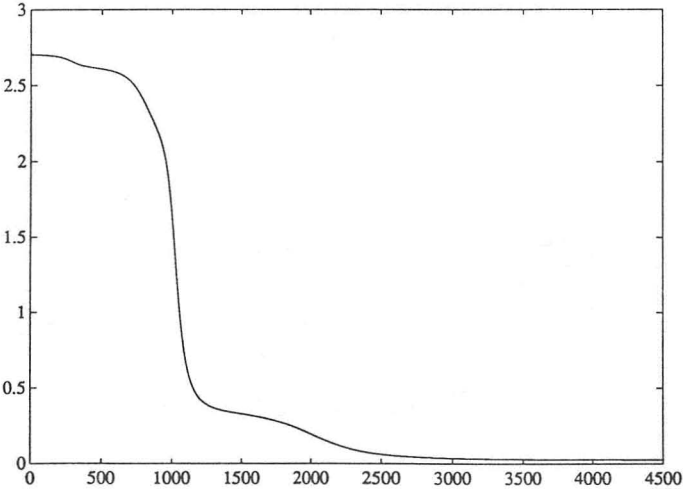

Figure 16: Square error for the BP algorithm.

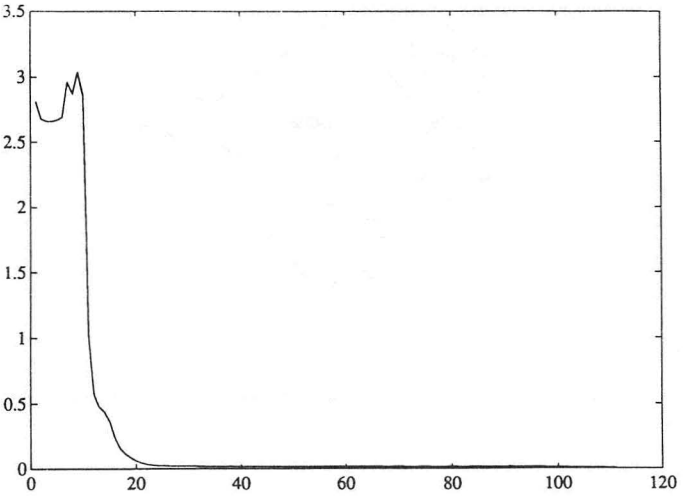

Figure 17: Square error for the GBP algorithm.

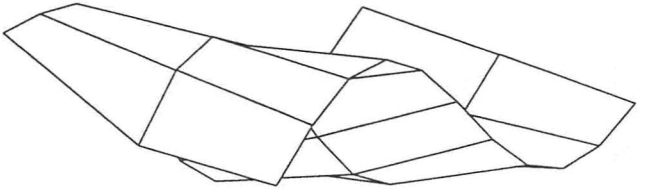

Figure 18: Initial surface.

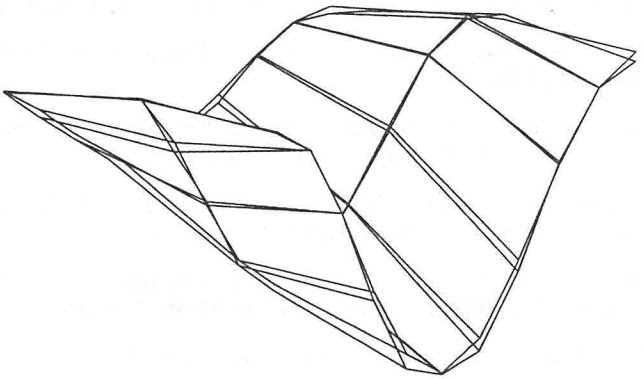

Figure 19: Final and reference surface.

and it is possible to have a great number of patterns (theoretically infinite) and only a small number are used to train the NN, so it is possible to choose an adequate training set for a smooth representation, giving a better behavior in terms of convergence.

A different situation arises in the discrete case, where it is possible to have some input patterns that do not differ very much from each other and yet produce a very different system output. In terms of our interpretations, this means a rapidly varying hypersurface and therefore a very hard surface to match with few components.

Further work is in progress to establish more clearly under precisely what conditions it is possible to get convergence to the required global solution.

References

[1] A.E. Albert and L.A. Gardner, *Stochastic Approximation and Nonlinear Regression* (MIT Press, Cambridge, MA, 1967).

[2] J.L. McClelland and D.E. Rumelhart, *Explorations in Parallel Distributed Processing* (MIT Press, Cambridge, MA 1988).

[3] T.J. Sejnowski, "Neural network learning algorithms," in *Neural Computers* (Springer-Verlag, 1988) 291–300.

[4] R. Hecht-Nilsen, "Neurocomputer applications," in *Neural Computers* (Springer-Verlag, 1988) 445–453.

[5] R. Meir, T. Grossman, and E. Domany, "Learning by choice of internal representations," *Complex Systems*, **2** (1988) 555–575.

[6] H. Bourland and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological Cybernetics*, **59** (1988) 291–294.

[7] A.E. Albert, *Regression and the Moore–Penrose Pseudoinverse* (Academic Press, New York, 1972).

[8] L.B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Computer*, **21** (1988) 25–39.

[9] D.S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, **2** (1988) 321–355.

[10] G.J. Mitchison and R.M. Durbin, "Bounds on the learning capacity of some multilayer networks," *Biological Cybernetics*, **60** (1989) 345–356.

[11] N.J. Nilsson, *Learning Machines* (McGraw-Hill, New York, 1974).

[12] M.L. Minsky and S.A. Papert, *Perceptrons* (MIT Press, Cambridge, MA, 1988).

[13] G. Josin, "Neural-space generalization of a topological transformation," *Biological Cybernetics*, **59** (1988) 283–290.

[14] W.F. Trench and B. Kolman, *Multivariable Calculus with Linear Algebra and Series* (Academic Press, New York, 1974).

[15] C. Moler, J. Little, and S. Bangert, *MATLAB User's Guide* (Mathworks Inc., 1987).

[16] L. Ljung, *Systems Identification — Theory for the User* (Prentice Hall, Englewood Cliffs, NJ, 1987).

[17] L. Ljung and T. Soderstrom, *Parameter Identification* (MIT Press, London, 1983).