Speed Up Learning and Network Optimization With Extended Back Propagation

Alessandro SPERDUTI, Antonina STARITA
Department of Computer Science
Corso Italia, 40 - 56100 - PISA
ITALY

Abstract-Methods to speed up learning in back propagation and to optimize the network architecture have been recently studied. This paper shows how adaptation of the steepness of the sigmoids during learning treats these two topics in a common framework. The adaptation of the steepness of the sigmoids is obtained by gradient descent. The resulting learning dynamics can be simulated by a standard network with fixed sigmoids and a learning rule whose main component is a gradient descent with adaptive learning parameters. A law linking variation on the weights to variation on the steepness of the sigmoids is discovered. Optimization of units is obtained by introducing a tendency to decay to zero in the steepness values. This decay corresponds to a decay of the sensitivity of the units. Units with low final sensitivity can be removed after a given transformation of the biases of the network. A decreasing initial distribution of the steepness values is suggested to obtain a good compromise between speed of learning and network optimization. Simulation of the proposed procedure has shown an improvement of the mean convergence rate with respect to the standard back propagation and good optimization performance. Several 4-3-1 networks for the four bits parity problem were discovered.

Key Words: Supervised learning, Back propagation, Speed-up learning, Network optimization.

1. INTRODUCTION

Several techniques have been studied to speed up back propagation (Rumelhart, Hinton, & Williams, 1986), such as the introduction of a momentum in the learning rule (Rumelhart, Hinton, & Williams, 1986), the use of alternative cost functions instead of the standard quadratic error function (Fahlman, 1988; Krogh, Thorbergsson, & Hertz, 1989; Solla, Levin, & Fleisher, 1988), and dynamic adaptation of the learning parameters (Cater, 1987; Jacobs, 1988; Jutten, Guerin, & Nguyen, 1991; Vogl, Mangis, Rigler, Zink, & Alkon, 1988). Methods to optimize the network architecture have also been developed. Optimization of network architecture is very important both to reduce the computational load and to improve generalization. Two different approaches have been proposed in literature to achieve this aim. In the pruning approach, a network is trained and then analyzed by unit activity (Mozer & Smolensky, 1989) or a relevance measure over weights (Karnin, 1990). Non relevant weights or units are subsequently removed by an iteration process which interleaves removing of weights/units with learning to recover the network damage. The other approach tries to force out pointless connections from the final network by introducing a tendency to decay to zero for each connection during learning (Chauvin, 1989; Hanson & Pratt, 1989; Hinton, 1986). The tendency to decay to zero is obtained by adding a term to the error function that penalizes the growth of the

weights. A recent work (Sietsma & Dow, 1991) combines these two approaches and introduces another technique that allows the network to be lengthened.

This paper is based on a very simple idea which treats the two trends discussed above, speed up learning and network optimization, in a common framework. We observe that back propagation is usually applied to networks with sigmoidal activation functions with fixed slope. In literature there is no specific work that justifies the choice to maintain the slope of the sigmoids fixed versus a dynamic adaptation of the steepness parameters, in addition to the adaptation of the weights. Perhaps the main reason for this choice is that neural networks have been conceived as a demonstration of how several interacting units, computing the same simple function, can perform complex computations. From this point of view, the differentiation of the computational abilities of a unit with respect to the others is only obtained by the differentiation of the weights on the connections entering the unit. Even if this is theoretically correct, some aspects about the efficiency of the learning process, such as the speed of the learning and the selection of the minimal network to implement the target function, are not considered.

In this paper we propose to extend the back propagation framework by adding a gradient descent on the steepness parameters of the sigmoids (Sperduti & Starita, 1991) and we show this extension provides not only an improvement of the mean convergence rate of the learning, but also the possibility to perform optimization over the number of units by adding a spontaneous tendency to decay to zero in the steepness values. In fact, the slope of the sigmoid of a unit can be considered as an index of the sensitivity of the unit to the inputs, and units that are not sensitive can be removed from the network because they are informationally useless.

This paper is organized as follows. In Section 2 we introduce gradient descent over the steepness parameters and we show this technique corresponds to use a network with fixed steepness parameters and a learning rule of the form:

$$(0)\quad \Delta\text{weight}_{jk} = -\mathbf{F}(\text{training set}, \text{weight}_{jk}) \frac{\partial \text{Error}}{\partial \text{weight}_{jk}} + \mathbf{G}(\text{training set}, \text{net}_k)\left(\frac{\partial \text{Error}}{\partial \text{weight}_{jk}}\right)^2,$$

where **F** and **G** are real functions whose main dependences are shown and net_k refers to the net input to the unit k. The first term of the right part of Eq. (0) is clearly a gradient descent with a dynamical learning rate **F**, whereas the second term is not easily understandable. Note that **F** depends on the training set and on the connections in consideration, thus the result is a different learning rate for each training pattern and for each connection in the network. This diversification of learning parameters has been recognized as useful in several papers that introduce the dynamical adaptation of the learning parameters (Cater, 1987; Jacobs, 1988; Vogl, Mangis, Rigler, Zink, & Alkon, 1988). The main difference between these works and our approach is that they use heuristic procedures to adapt the learning parameters, meanwhile we obtain the adaptation of the learning parameters through a total error minimization procedure, thus in a theoretically more rigorous way. A minimization procedure has also been introduced by Jutten et al. (1991), but since it requires a learning parameter for each connection, the complexity of the learning procedure is higher than ours.

Section 2 also presents a law linking variation on the weights to variation on the steepness parameters which is then used in the successive Section.

In Section 3 we show that the introduction of variation on steepness parameters means that architecture can be optimized with respect to the number of units. The relationship between the steepness value and the sensitivity of a unit is formally stated in two forms: in the first, sensitivity is defined independently from the training set, in the second it is

defined according to the training set. Optimization of units is then obtained by introducing a tendency to decay to zero in the steepness values and showing this corresponds to a decay of the sensitivity of the units. In this way, units which are not useful in implementing the target function get their sensitivity to zero and can be removed after a given transformation of the biases of the network. Since the decay on the steepness parameters slows the learning, a particular initial distribution for the steepness parameters is discussed at the end of the Section. This distribution only assigns initial significative steepness values to a subset S of the hidden units. The size of S is set to the supposed minimal number of hidden units needed to implement the target function. If the size of S is too small for the real needs of the target function, the learning on the steepness parameters will recruit a new hidden unit by increasing its steepness value. A small decay in the steepness values is maintained to avoid the growth of the steepness values of units which are not needed to implement the target function. The resulting procedure is thus a compromise between speed of learning and network optimization.
In Section 4 we present the results of simulations on two problems: the four bits parity problem, and 8x8 dot numeric font learning. Results about the speed up of the learning with respect to standard back propagation are presented for both the problems. Whereas, results about units optimization are presented for the first problem.
A discussion about the proposed extention of the back propagation and its relations with other works is then presented in Section 5.

2. LEARNING RULE FOR STEEPNESS PARAMETERS

The back propagation algorithm is based on the gradient descent rule:

$$(1) \qquad \Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}},$$

where w_{jk} is the weight on the connection outgoing from the unit j and entering the unit k, and η is the learning rate. In the standard back propagation framework η is constant and E is defined as:

$$(2) \qquad E = \frac{1}{2} \sum_p \sum_j (t_j^p - O_j^p)^2,$$

where t and O are respectively the target and the actual network output, the index p varies on the training set, and the index j varies on the output units. In the back propagation framework each computational unit computes the same activation function. The activation function is normally a sigmoid function chosen between $\frac{1}{1 + \exp(-\beta\ net)}$ and $\tanh(\beta\ net)$. The steepness parameter β is often set to 1 and not changed by the learning rule. Our proposal is to perform gradient descent on the steepness parameters as well. In our framework each unit k has an associated steepness parameter β_k which is changed by the rule:

$$(3) \qquad \Delta\beta_k = -\varepsilon \frac{\partial E}{\partial \beta_k}.$$

In the appendix we show how to derive the explicit rule for $\Delta\beta_k$ when the function $\frac{1}{1 + \exp(-\beta_k \text{ net})}$ is used as an activation function by the k unit.

2.1 Learning Dynamics

The resulting network learning dynamics can be understood by noting that the net term of the activation function times the steepness parameter can be rewritten in the standard framework (steepness parameter equals one) as follows[1]:

$$(4) \qquad \beta_k \, net_k^p = \beta_k \sum_j w_{jk} I_j^p = \sum_j \beta_k w_{jk} I_j^p = \sum_j z_{jk} I_j^p \, ,$$

where I^p is the input vector to the neuron k generated by the presentation of the training pattern p to the network, and $z_{jk} = \beta_k w_{jk}$. Eq. (4) allows us to study learning dynamics in a network with steepness parameters which equal one. Since $z_{jk} = \beta_k w_{jk}$, at each iteration of the learning rule we have:

$$\begin{aligned} (5) \qquad z_{jk}^{new} &= \beta_k^{new} \, w_{jk}^{new} \\ &= (\beta_k^{old} + \Delta\beta_k)(w_{jk}^{old} + \Delta w_{jk}) \\ &= \beta_k^{old} \, w_{jk}^{old} + \Delta\beta_k w_{jk}^{old} + \beta_k^{old} \, \Delta w_{jk} + \Delta\beta_k \, \Delta w_{jk} \\ &= z_{jk}^{old} + \Delta\beta_k w_{jk}^{old} + \beta_k^{old} \, \Delta w_{jk} + \Delta\beta_k \, \Delta w_{jk}. \end{aligned}$$

Eq. (5) gives the learning rule for z_{jk}:

$$(6) \qquad \Delta z_{jk} = \Delta\beta_k w_{jk}^{old} + \beta_k^{old} \, \Delta w_{jk} + \Delta\beta_k \, \Delta w_{jk}.$$

It is possible to rewrite eq. (6) by using the partial derivative of the error function with respect to z_{jk}. To obtain such a form of the (6) we need to express Δw_{jk} and $\Delta\beta_k$ in a more suitable form. First of all note that since

$$(7) \qquad \frac{\partial z_{jk}}{\partial w_{jk}} = \beta_k,$$

the weights w_{jk} are changed according to the equation:

$$(8) \qquad \Delta w_{jk} = -\eta \, \frac{\partial E}{\partial w_{jk}} = -\eta \, \frac{\partial E}{\partial z_{jk}} \frac{\partial z_{jk}}{\partial w_{jk}} = -\eta \, \beta_k \frac{\partial E}{\partial z_{jk}} \, .$$

---

[1] We use the convention that $\forall p$, $I_0^p = 1$; thus the weight w_{0k} represents the bias for the k unit.

A similar equation for $\Delta\beta_k$ can be derived if we consider an input pattern I^p at time. Let us define the net input to the neuron k in the standard framework, given a pattern I^p in input, as:

$$(9) \qquad netz_k^p = \sum_j z_{jk} I_j^p = \beta_k net_k^p \ .$$

The $netz_k^p$ term is very useful for our purpose. The first equation we need is:

$$(10) \qquad \frac{\partial netz_k^p}{\partial \beta_k} = net_k^p .$$

Then we can derive a second equation

$$(11) \qquad \frac{\partial E}{\partial z_{jk}} = \frac{\partial E}{\partial netz_k^p} \frac{\partial netz_k^p}{\partial z_{jk}} = \frac{\partial E}{\partial netz_k^p} I_j^p \ ,$$

which allows us to express the partial derivative of the error function of the $netz_k^p$ term by using the partial derivative of the error function of z_{jk}:

$$(12) \qquad \frac{\partial E}{\partial netz_k^p} = \frac{1}{I_j^p} \frac{\partial E}{\partial z_{jk}} \ ,$$

where $I_j^p \neq 0$.
Given an input pattern p, eq. (10) and (12) together with eq. (3) lead to the following learning rule for the steepness parameters:

$$(13) \qquad \Delta\beta_k = -\varepsilon \frac{\partial E}{\partial \beta_k} = -\varepsilon \frac{\partial E}{\partial netz_k^p} \frac{\partial netz_k^p}{\partial \beta_k} = -\varepsilon \frac{net_k^p}{I_j^p} \frac{\partial E}{\partial z_{jk}} \ ,$$

where $I_j^p \neq 0$, and $\Delta\beta_k$ is referred to the input pattern p. Using equations (6)-(13) we obtain, by omitting the old label :

$$(14) \qquad \Delta z_{jk} = w_{jk} \Delta\beta_k + \beta_k \Delta w_{jk} + \Delta w_{jk} \Delta\beta_k$$

$$= - \varepsilon \frac{net_k^p}{I_j^p} \frac{\partial E}{\partial z_{jk}} \; w_{jk} - \eta \; \beta_k^2 \frac{\partial E}{\partial z_{jk}} + \eta\varepsilon \frac{net_k^p}{I_j^p} \beta_k \left(\frac{\partial E}{\partial z_{jk}}\right)^2$$

$$= - \frac{\partial E}{\partial z_{jk}} \left(\varepsilon \frac{net_k^p}{I_j^p} \; w_{jk} + \eta \; \beta_k^2 \right) + \eta\varepsilon \frac{netz_k^p}{I_j^p} \left(\frac{\partial E}{\partial z_{jk}}\right)^2 ,$$

where $I_j^p \neq 0$, and $\Delta\beta_k$, Δw_{jk}, and Δz_{jk} are referred to the input pattern p.
Equation (14) describes the dynamics of the learning for a single input pattern. The learning behaviour 'by epoch' is given by summing the Δz_{jk} terms obtained from each pattern in the training set. As eq. (14) shows, the learning is obtained by a gradient descent with dynamic learning rate:

$$(15) \qquad \tau_{jk} = \varepsilon \frac{net_k^p}{I_j^p} \; w_{jk} + \eta \; \beta_k^2 ,$$

plus a quadratic term of the partial derivative of the error. This dynamic is not easily interpretable. What we can observe is that (15) depends on how z_{jk} is decomposed in w_{jk} and β_k, and that the quadratic term is independent from this decomposition.

Note how the dynamic learning rate τ_{jk} depends on the training pattern I_j^p and on the weight on the connection jk. Because of this dependence, there is a different learning rate for each training pattern and for each connection.

2.2 Relationship between Δw_{jk} and $\Delta\beta_k$

Equations (8) and (13) can be used to derive a relationship between the changes in w_{jk} and the changes in β_k. In fact, rewriting in (8) the derivative of the error function of z_{jk} in terms of $\Delta\beta_k$ as prescribed by eq. (13) we obtain:

$$(16) \qquad \Delta w_{jk} = \frac{\eta}{\varepsilon} \frac{I_j^p}{net_k^p} \beta_k \Delta\beta_k ,$$

where $I_j^p \neq 0$, $net_k^p \neq 0$, $\Delta\beta_k$ and Δw_{jk} refer to the input pattern p. Using eq. (16) it is possible to rediscover a relationship from the standard back propagation framework and which is computationally very useful:

$$(17) \qquad \forall \; i,j \in INP(k), \; \Delta w_{ik} = \frac{I_i^p}{I_j^p} \Delta w_{jk},$$

where $I_j^p \neq 0$, $\Delta w_{jk} \neq 0$, and INP(k) is the set of indexes of the neurons from which the neuron k receives an input. Equations (16) and (17) can be used to build an efficient algorithm. This algorithm computes the $\Delta\beta_k$ as shown in Appendix and then the weights changes using eq. (16) for one j in INP(k) and eq. (17) for the remaining indexes in INP(k).

3. ARCHITECTURE OPTIMIZATION

One approach to reduce the number of connections and/or the number of units in a solution network is to give on weights on the connections a tendency to decay to zero during learning. This solution was proposed by several authors (Hinton, 1986; Scalettar & Zee, 1988; Kramer & Sangiovanni-Vincentelli, 1989). In particular, optimization on the number of hidden units was obtained by making the weight decay rates larger for units that have small outputs or small incoming weights (Chauvin, 1989; Hanson & Pratt, 1989). In this section we will show that optimization on the number of units can be performed very naturally in our framework.
First of all we need to state some concepts to understand how optimization works, after we will discuss the introduction of a decay on the steepness parameters with these concepts.

3.1 Sensitivity of a Unit and Reduction of the Network

In the standard framework the input to a unit k is given by the dot product of the input pattern I^p with the weights vector W_k of the unit:

$$(18) \quad \text{input}(k) = W_k \cdot I^p = \| W_k \| \, \| I^p \| \cos(\phi_{pk}),$$

where ϕ_{pk} is the angle between I^p and W_k. If the components of I^p can be bounded in module by 1, as is the case for units which receive the input from other units in the network when sigmoidal activation function like $\frac{1}{1 + \exp(-net)}$ and/or $\tanh(net)$ are used[2], then $\| I^p \| \leq \sqrt{n}$, where n is the dimension of the I^p vector, and input (k) can be bounded by:

$$(19) \quad |\text{input}(k)| \leq \| W_k \| \sqrt{n} = \sigma(k),$$

since $|\cos(\phi_{pk})|$ is always less than 1.
In general $\sigma()$ can be used as an index of the *sensitivity* of a unit to its input. If $\sigma(k)$ is very small, the unit k can be considered insensitive to the input. Thus the unit k can be removed by the network if the biases of all the units in OUT(k), the set of units receiving the output from unit k, are modified according to the weights which leave unit k:

$$(20) \quad \forall i \in \text{OUT}(k), \; w_{0i}^{new} = w_{0i}^{old} + o_k \, w_{ki} \, ,$$

---

[2]Note that the training patterns can be recoded so as to satisfy the constraint on the module.

where the coefficient o_k is the output of the activation function computed in zero.
If the distribution of the training patterns is such that:

(21) $\forall\ p,\ \| I^p \| \ |\cos(\phi_{pk})| \leq \xi_k \ll \sqrt{n}$,

then it is advantageous to define the *sensitivity (of a unit) with respect to the training set P*:

(22) $$\sigma_{|P}(k) = \frac{\sigma(k)\ \xi_k}{\sqrt{n}}.$$

It is obvious that the reduction in the original network by using $\sigma()$ preserves the function computed by it better than $\sigma_{|P}()$ because it does not depend on the training set P. The nearer ξ_k is to zero, the higher the probability to modify the function by a $\sigma_{|P}()$ reduction. However the two reductions are functionally equivalent over the training set P. The advantage to using $\sigma_{|P}()$ is that more units may be pruned.
Condition (21) is a particular instance of a more general case:

(23) $\forall\ p,\ |\ \| I^p \| \cos(\phi_{pk}) - \nu_k| \leq \xi_k$,

where ν_k is a constant. In this case the reduction rule (20) must be applied with o_k computed in ν_k and is the same as presented in (Sietsma & Dow, 1991), with the difference that we analyze the input of unit instead of the output. However our definition of $\sigma()$ is applicable without considering the training set P.

3.2 Decay of the Steepness Parameters

In our framework the sensitivity $\sigma(k)$ becomes:

(24) $\sigma(k) = |\beta_k|\ \| W_k \| \sqrt{n}$,

and the optimization on the number of the units in the network can be performed in a very natural fashion by giving to the steepness parameters β_k a tendency to decay to zero. The decay of the steepness parameters will correspond to a decay in the sensitivity of the units as shown by eq. (24). In this paper we restrict the steepness parameters to positive values. This restriction was introduced for two reasons. Firstly, it reduces the search space without excluding solutions since a solution with negative steepness parameters can be transformed into a solution with positive steepness parameters by changing the sign to the weights entering a unit with negative steepness parameter. Secondly, in unconstrained simulations the steepness parameter of some output unit may become null during a transition from positive to negative values (or viceversa), rendering the unit insensitive to the input stimulus.

Since $\sigma()$ depends on $||\ W_k||$ as well, and the decay of β_k could be balanced by an increase in $||\ W_k||$, it is interesting to study how $||\ W_k||$ changes during learning on one pattern p:

$$
\begin{aligned}
(25) \qquad ||\ W_k^{new}||^2 &= \sum_i (w_{ik}^{old} + \Delta w_{ik})^2 \\
&= \sum_i (w_{ik}^{old})^2 + \sum_i (\Delta w_{ik})^2 + 2\sum_i (w_{ik}^{old} \Delta w_{ik}) \\
&= ||\ W_k^{old}||^2 + \sum_i (\frac{\eta}{\varepsilon} \frac{I_i^p}{net_k^p} \beta_k \Delta\beta_k)^2 + 2\frac{\eta}{\varepsilon} \frac{\beta_k \Delta\beta_k}{net_k^p} \sum_i (I_i^p w_{ik}^{old}) \\
&= ||\ W_k^{old}||^2 + ||\ I^p\ ||^2 (\frac{\eta}{\varepsilon} \frac{\beta_k \Delta\beta_k}{net_k^p})^2 + 2\frac{\eta}{\varepsilon} \beta_k \Delta\beta_k \\
&= ||\ W_k^{old}||^2 + ||\ I^p\ ||^2 (\frac{\eta}{\varepsilon} \frac{\beta_k \Delta\beta_k}{||W_k^{old}||\ ||I^p|| cos(\phi_{pk})})^2 + 2\frac{\eta}{\varepsilon} \beta_k \Delta\beta_k \\
&= ||\ W_k^{old}||^2 + (\frac{\eta}{\varepsilon} \frac{\beta_k \Delta\beta_k}{||W_k^{old}|| cos(\phi_{pk})})^2 + 2\frac{\eta}{\varepsilon} \beta_k \Delta\beta_k\ ,
\end{aligned}
$$

where we used eq. (16) (remind $I_j^p \neq 0$, $net_k^p \neq 0$).

Equation (25) shows that, given $\beta_k > 0$, $||\ W_k^{new}||$ will be lower than $||\ W_k^{old}||$ if $\Delta\beta_k$ is such that:

$$
(26) \qquad -2(\frac{\varepsilon}{\eta} \frac{||W_k^{old}||^2 cos(\phi_{pk})^2}{\beta_k}) < \Delta\beta_k < 0,
$$

otherwise $||\ W_k^{new}||$ will increase.

Since β_k gathers global information on the output of the k unit, it is not reasonable that $||\ W_k^{new}||$ increases when β_k decreases and viceversa. If this happens, it means that the two components of the learning, learning on the weights and learning on the steepness parameters, are working against each other. Thus, to avoid pointless computations, condition (26) must be satisfied when $\Delta\beta_k$ is negative.

The probability that a negative $\Delta\beta_k$ satisfies condition (26) can be increased by tuning the learning parameters. Setting $\frac{\varepsilon}{\eta}$ to be large should reduce the variability of $||\ W_k||$ and increase the dependence of $\sigma(k)$ on β_k, however an η which is too small can seriously degrade the learning performances.

In conclusion, a decay in the steepness parameters optimizes the network architecture introducing a decay on the sensitivity of useless units. The amount of decay in sensitivity

depends on $\frac{\varepsilon}{\eta}$, and care must be taken to preserve the ability of the learning to discover a solution.
We obtain the tendency to decay to zero for the steepness parameters by adding the following term to the error function:

$$(27) \qquad E_d = \frac{\gamma}{2} \sum_j \frac{\beta_j^2}{1+\beta_j^2},$$

where the index j varies over all the neurons in the network and γ is the decay rate. The E_d term was chosen to allow slow decay for large steepness parameters and more rapid decay for small steepness parameters. In this way a large steepness parameter is not penalized over several small steepness parameters.
The change in the steepness parameters obtained by adding (27) to the error function is:

$$(28) \qquad \Delta\beta_k = -\varepsilon \frac{\partial(E+E_d)}{\partial\beta_k} = -\varepsilon \frac{\partial E}{\partial\beta_k} - \varepsilon \frac{\gamma \beta_k}{(1+\beta_k^2)^2}.$$

The final learning rule for the steepness parameters becomes:

$$(29) \qquad \beta_k^{new} = \beta_k^{old} \left(1 - \varepsilon \frac{\gamma}{(1+\beta_k^2)^2}\right) - \varepsilon \frac{\partial E}{\partial\beta_k}.$$

At the end of the learning, the $\sigma()$ (or $\sigma_{|p}()$) values are computed for each unit and the units with low value are removed according to eq. (20).
The advantage of our framework is that by only considering the steepness parameters to optimize the number of hidden units, the computational expense for a full connected feed-forward network with N units is reduced from $O(N^2)$ to $O(N)$.

3.3 Recruitment of Hidden Units

One drawback of the decay technique is that it is difficult to set γ in such a way as to obtain good results without risking slowing down the convergence speed of the learning or causing all the steepness parameters to be null. We thus tested a particular initial distribution of the steepness parameters where only one subset S of the hidden units is initialized with a significant steepness value; all the other units are initialized with a small steepness value. The idea is to have a decay which can inhibit the activity of units not in S which are not needed to approximate the target function, but not strong enough to damage the learning. The size of S can be decided by estimating the minimal number of hidden units needed to approximate the target function. Moreover we suggest distributing the steepness values to the units in the S set according to a decreasing function (see Fig. 1). This non-uniform distribution has two goals. Firstly, it breaks down the symmetries of the search space and speeds up learning. Secondly, if the size of S is bigger than the

minimum needed, it is more probable that units with small steepness values are not involved in the learning and so it is easier to discard them from the final network.
If the chosen size of S is smaller than required by the computation, the learning will recruit one or more units from those not in S, thus overcoming the decay on their steepness parameters.

4. SIMULATION

We made two simulations to test our algorithm versus the standard back propagation. We chose the four bits parity problem (Subsection 4.1), and 8x8 dot numeric font learning (Subsection 4.2).
In the four bits parity problem we added a momentum term to the weights change. The final correction of the weights is described as

$$(30) \quad \Delta w_{jk}^{new} = -\eta \frac{\partial E}{\partial w_{jk}^{old}} + \alpha \Delta w_{jk}^{old},$$

where α is the momentum constant.
We used the extended generalized delta rule as shown in Appendix A, and we updated the network parameters by epoch to approximate the gradient of the error function better. In these simulations the steepness parameters are bounded to positive values greater than 0.05.
In the last Subsection 4.3 results on the optimization of hidden units for the four bits parity problem are presented and discussed.

4.1 Four bits parity problem

The parity problem is a difficult problem because of its non-linearity. We used a 4-6-1 feedforward network and stopped the training when the total error E decreased to less than 0.001 or the number of epochs was more than 5000. Ten random starting points in the weight space were tested for two different distributions of the steepness parameters. The first steepness distribution (SD1) sets all the parameters to 1.0, the second one (SD2) sets the steepness parameters of the hidden units as shown in Fig. 2 and the parameter for the output unit to a random value in the range [0.5,1.5] for each starting point.
Figure 3 shows the mean learning curves for the two different initial distributions of the steepness parameters, both for the standard (d and b) and the extended learning procedure (c and a). Standard learning is obtained by setting ε to zero. The learning parameters values are reported at the top of the figure. To compute the mean learning curves, the error of trials converging before the 5000-th epoch is fixed to the last computed value.
The mean learning curves show, in any case, that extended back propagation performs better if high precision in the output value is required. Even if the (c) curve (obtained with the SD1 distribution) is worse than the (d) curve before the 3000-th epoch, this is due to a few tests which had a late convergence. However, a general improvement in the learning is clearly obtained by the extended back propagation if the SD2 distribution is used (curve (a)). Note that in this case all the tests got a global minimum. Local minima, on the other hand, were found in the standard learning (b found 2 local minima and d found 3 local minima) and in the extended learning with SD1 (1 local minimum).

4.2 8x8 dot numeric font learning

To solve the 8x8 dot numeric font learning we used a larger network than the 4-6-1 one: a 64-6-10 network, where each output unit is associated with an input pattern by which it is activated and each input unit corresponds to a dot in the 8x8 pattern grid. The training set is shown in Fig. 4. When the dot is white (black), zero (one) is input to the corresponding input unit.
The learning was stopped when the error E decreased to less than 0.001, or the number of epochs was more than 2000. Five random starting points were considered. No momentum was added because preliminary simulations gave inferior performances.
Several values of η were tested for the original back propagation algorithm and steepness distribution SD2. Figure 5 shows the resulting mean curves. The best performances are obtained when η is set to 1.5.
Extended back propagation was then tested using η at 1.5 and increasing values of ε. The mean learning curves are shown in Fig. 6. The learning shows an improvement for values of ε less than or equal to 0.6. Larger values of ε lead to instabilities in the learning process.
These simulations show that the extended procedure doesn't only correspond to a scaling in the learning parameters with respect to the standard procedure, but it really improves the learning.

4.3 Units Optimization

Optimization on hidden units was tested for the four bits parity problem (Section 4.1) following what was described in Sections 3.2 and 3.3. Twenty random starting points in the weight space were tested with initial steepness distribution in Fig. 7. The first four hidden units had a significant steepness value and constituted the S set (see Section 3.3). The cardinality of the S set was chosen as four because four is the minimal number of hyperplanes in the weight space needed to solve the problem when linear thresholds are used (see Venkatesh & Baldi, 1991).
The steepness values of the units were changed according to Eq. (27), which includes a spontaneous decay in the steepness values (see Section 3.2).
A first set of simulations was performed with learning parameters $\mu = 0.9$, $\eta = \varepsilon = 0.3$, and $\gamma = 0.001$. With this setting the ratio $\frac{\varepsilon}{\eta}$ is one. The stop criteria were as in Section 4.1. At the end of the learning units with sensitivity ($\sigma()$) value under 0.1 can be removed from the network by making the transformation discussed in Eq. (20). The results of the simulations are reported in Fig. 8, where the patterns of the bars show how many units can been removed from the final solution, and the height of the bars is proportional to the velocity of the learning. In three cases the network was unable to learn the parity function in 5000 epochs (corresponding bars are filled with the failure pattern). One of these has shown instabilities during learning (E = 7.99984 after 5000 epochs). In the other two cases the learning may either be trapped in local minima or in a flat region of the error surface. Note that in several cases the number of hidden units can be reduced to four, i.e. the supposed minimal number of hidden units needed to implement the parity function.

A second set of simulation was performed with the same previous values for μ and γ but $\eta = 0.05$, and $\varepsilon = 0.5$. In this case the ratio $\frac{\varepsilon}{\eta}$ is 0.1 and the dependence of the sensitivity $\sigma()$ on the steepness learning should increase as discussed in Section 3.2.
The results of the simulations are reported in Fig. 9, using the same representations of Fig. 8. Note the different scale on the ordinates. The learning is slower, but a slightly better optimization on the network is obtained. The final solution of sample 10 can be reduced by three hidden units, showing that our evaluation of the minimal number of hidden units was wrong. This solution is also a result of the reduced value of the ratio $\frac{\varepsilon}{\eta}$.
The number of cases in which the network was not able to learn is comparable to the previous one (4 cases versus 3 cases).
To test the ability of the extended back propagation to learn if the cardinality of the S set is smaller than the minimal number of hidden units needed by the target function, the same simulations presented above were repeated with a different initial distribution of the steepness parameters. The distribution used is shown in Fig. 10 and is characterized by the fact that only two hidden units have significant steepness value (i.e., belong to S). The results of the simulations are shown respectively in Figs. 11 ($\frac{\varepsilon}{\eta}$=1) and 12 ($\frac{\varepsilon}{\eta}$ = 0.1).
In the first set of simulations the learning has trouble in converging and optimization is not particularly good. In the second set of simulations the learning seems to converge with more probability (despite being slower) and the hidden units are optimized very well.
We have reported the number of units that can be optimized a priori, i.e., units with very little sensitivity. These units can be removed without looking at the values of the outgoing connections. In the set of simulations shown in Figs. 9 and 11 sometimes one more hidden unit (with small sensitivity value) can be optimized since the weight on its outgoing connection was small. However a hidden unit generally has several outgoing connections and the probability to remove it becomes lower.
Figure 13 shows the optimized network of sample 6 reported in Fig. 12.
To show that the combination of the decay technique with the initial decrescent distribution of the steepness parameters causes complex behaviour in the learning, in Figs. 14 and 15 we have reported how the sensitivity of the hidden units changes with respect to the γ parameter in two convergent cases. The first case (sample 12) the sensitivity of the hidden units is lowered homogeneously by increasing the decay parameter γ. In the second case (sample 1) the sensitivity has a more complex behaviour depending on the hidden unit in consideration. Our interpretation of the results obtained with sample 1 is that a different global minimum was discovered with the highest value of γ.
The complexity of the learning dynamics is also shown by the fact that often one of the optimized units belongs to S.
Optimization using the sensitivity defined with respect to the training set ($\sigma_{|P}()$) gave the same results. We think this is due to the completeness of the training set. More work must be done to establish the conditions for which $\sigma_{|P}()$ works better than $\sigma()$.

5. CONCLUSIONS

When back propagation (BP) was developed, the main aim was to demonstrate a procedure which could derive a set of weights for a multi-layer network. Today the main aim is to have a reliable and efficient tool to demonstrate the applicability of artificial neural networks to real problems. Extended back propagation (BP^+) is a framework in which learning speed up and network optimization can be treated in a unified way. BP is a particular instance of this framework ($\varepsilon=0$) and some recent work on BP can be revisited from this point of view. For example Rigler et al. (1991) proposed rescaling BP variables. This is justified by the particular form of the logistic derivative. From the BP^+ point of view this rescaling is the effect of the minimization of the error in relation to the steepness parameters (see Appendix, eqs. A5-8). The adaptation of the learning parameters is also introduced in the BP^+ framework. Equation (15) shows that both the classes of learning parameters suggested by Cater (1987) (η^p, one for each pattern p) and Jacobs (1988) (η^{ik}, for each connection ik) are implicitly in the BP^+. In fact, τ_{ik} depends both on the input pattern and on the weight on the connection ik. Such implicit learning parameters are adapted according to the initial settings of the weights and steepness parameters, and to minimization of the total error E. How important choosing the initial setting is, has been demonstrated by the results of the four bits parity problem, when the SD2 distribution was used.

Another advantage of the BP^+ framework is that other accelerating techniques, such as momentum, alternative cost functions, and expected source values (Samad, 1991), are allowed inside it without modifications.

On the optimization side, the definition of the sensitivity of a unit given in Section 3.2 is particular useful because it can remove units independently of the training set. Connections can be removed by considering the weight on the connection times the sensitivity of the unit which originate the connection. In addition the complexity of the optimization is reduced from $O(n^2)$ to O(n), since it involves units and not connections.

Last but not least, the computational complexity of the framework is close to the complexity of BP since the information needed to compute variations on the steepness parameters are the same needed in BP (see Appendix). The computational load added by the BP^+ is only due to the maintainance of a variable for each unit (the steepness value), thus it is O(n).

REFERENCES

Cater, J. P. (1987). Successfully Using Peak Learning Rates of 10 (and Greater) in Back-Propagation Networks with the Heuristic Learning Algorithm. *IEEE First International Conference on Neural Networks*, **11**,645-651.

Chauvin, Y. (1989). A Back-Propagation Algorithm with Optimal Use of Hidden Units. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems I* (pp. 519-526). San Mateo, CA: Morgan Kaufmann.

Fahlman, S. E. (1988). Fast-Learning Variations on Back-Propagation: An Empirical Study. In D. Touretzky, G. Hinton, T. Sejnowski (Ed.), *Proceedings of the 1988*

Connectionist Models Summer School (pp. 38-51). San Mateo, CA: Morgan Kaufmann.

Hanson, S.J., & Pratt L. (1989). A Comparison od Different Biases for Minimal Network Construction with Back-Propagation. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems I* (pp. 177-185). San Mateo, CA: Morgan Kaufmann.

Hinton, G. E. (1986). Learning Distributed Representations of Concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (pp. 1-12). Hillsdale: Erlbaum.

Jacobs, R. A. (1988). Increased Rates of Convergence Through Learning Rate Adaptation. Neural Networks,**1**, 295-307.

Jutten, C., Guerin, A., & Nguyen Thi, H.L. (1991). Adaptive Optimization of Neural Algorithms. In LNCS 540, A. Prieto (Ed.), *Artificial Neural Networks* (pp.54-61). New York: Springer-Verlag.

Karnin, E. D. (1990). A Simple Procedure for Pruning Back-Propagation Trained Neural Networks. In *IEEE Transactions on Neural Networks*, **1**, 239-242.

Kramer, A. H., & Sangiovanni-Vincentelli, A. (1989). Efficient Parallel Learning Algorithms for Neural Networks. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems I* (pp. 40-48). San Mateo, CA: Morgan Kaufmann.

Krogh, A., Thorbergsson, G. I., & Hertz J.A. (1989). A Cost Function for Internal Representations. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems II* (pp. 733-740). San Mateo, CA: Morgan Kaufmann.

Mozer, M. C., & Smolensky P. (1989). Using Relevance to Reduce Network Size Automatically. *Connection Science*, **1**, 3-16.

Rigler, A. K. , Irvine J. M. , Vogl T.P. (1991). Rescaling of Variables in Back Propagation Learning. *Neural Networks*, **4**, 225-230.

Rumelhart, D. E., Hinton, G.E., & Williams R.J. (Eds.). (1986). Learning internal representations by error propagation. In *Parallel distributed processing* (chap. 8, pp. 318-362). Cambridge, MA: MIT Press.

Samad, T. (1991). Back Propagation With Expected Source Values. *Neural Networks*, **4**, 615-618.

Scalettar, R., & Zee, A. (1988). Emergence of Grandmother Memory in Feed Forward Networks: Learning with Noise and Forgetfulness. In D. Waltz, J.A. Feldman (Ed.), *Connectionist Models and Their Implications: Readings from Cognitive Science* (pp. 309-332). Norwood: Ablex.

Sietsma, J., & Dow, R. J. F. (1991). Creating Artificial Neural Networks That Generalize. *Neural Networks*, **4**, 67-79.

Solla, S.A., Levin, E., & Fleisher, M. (1988). Accelerated Learning in Layered Neural Networks. *Complex Systems*, **2**, 625-639.

Sperduti, A. & Starita, A. (1991). Extension of Generalized Delta Rule to Adapt Sigmoid Functions. In *Proceedings of the 13th Annual International Conference IEEE/EMBS* (pp. 1393-1394). New York: IEEE Press.

Venkatesh, S. S., & Baldi, P. (1991). Programmed Interactions in Higher-Order Neural Networks: Maximal Capacity. *Journal of Complexity*, **7**, 316-337.

Vogl, T.P., Mangis, J. K., Rigler, A.K., Zink, W.T., & Alkon D.L. (1988). Accelerating the Convergence of the Back-Propagation Method. *Biological Cybernetics*, **59**, 257-263.

APPENDIX

For the sake of simplicity the extension of the back propagation on a feed-forward network with one hidden layer will be discussed. The extension to networks with any number of hidden layers is straightforward.

Let N_I, N_H, N_O be the number of input, hidden and output units of the network; let I^p be the input units state vector, i.e. the input pattern p; H^p and O^p be the hidden and output units state vector for the pattern p and f_j^p and y_j^p be the resulting potentials of the j-th hidden and output unit ; let w_{ij}^i be the weight from input unit i to hidden unit j and w_{ij}^h be the weight from hidden unit i to output unit j ; finally let the parametric sigmoid functions for the k-th hidden and output unit be defined as follows:

$$\text{(A1)} \quad FH_k(x) = (1 + e^{-\beta_k^h x})^{-1},$$

$$\text{(A2)} \quad FO_k(x) = (1 + e^{-\beta_k^o x})^{-1},$$

where β_k^h and β_k^o are respectively parameters of the k-th hidden and output unit.

In the network the states of the hidden and output units are computed by the equations:

$$\text{(A3)} \quad H_j^p = FH_j^p(\phi_j^p), \text{ where } \phi_j^p = \sum_{i=0}^{N_I} w_{ij}^i I_i^p,$$

$$\text{(A4)} \quad O_k^p = FO_k^p(\psi_k^p), \text{ where } \psi_k^p = \sum_{j=0}^{N_H} w_{jk}^h H_j^p,$$

Note that the unit with index zero in the input and hidden layers represents a constant bias ($I^p_0 = H^p_0 = 1$) which is connected to each unit, regardless of the pattern being processed.

The standard error function defined over the target output t^p and the network output O^p depends on the network weights and on the β^o_k and β^h_k parameters as well. In our learning schema, the standard back propagation rules regarding the changes of the network weights are retained:

(A5) $$\Delta w^h_{ij} = \eta \sum_p (t^p_j - O^p_j) FO^{p\,'}_j H^p_i \text{, where}$$

(A6) $$FO^{p\,'}_j = FO^{p\,'}_j(\psi^p_j) = \beta^o_j FO^p_j (1-FO^p_j) \text{, and}$$

(A7) $$\Delta w^i_{lk} = \eta \sum_p \sum_j (t^p_j - O^p_j) FO^{p\,'}_j w^h_{kj} FH^{p\,'}_k I^p_l \text{, where}$$

(A8) $$FH^{p\,'}_k = FH^{p\,'}_k(\phi^p_k) = \beta^h_k FH^p_k (1-FH^p_k) .$$

Moreover we propose to change the sigmoid functions parameters in the same way as back propagation changes the weights. The changes in the parameter β^o_k for the k-th output unit are:

(A9) $$\Delta\beta^o_k = -\frac{\eta}{2} \sum_p \sum_j \frac{\partial (t^p_j - O^p_j)^2}{\partial \beta^o_k} = \eta \sum_p \sum_j \frac{\partial O^p_j}{\partial \beta^o_k} (t^p_j - O^p_j) =$$

$$= \eta \sum_p \sum_j \frac{\partial FO^p_j}{\partial \beta^o_k} (t^p_j - O^p_j) = \eta \sum_p \frac{\partial FO^p_k}{\partial \beta^o_k} (t^p_k - O^p_k) ,$$

where the last equality is obtained because the partial derivative of FO^p_j is zero for all but the j = k term in the summation over j. The partial derivative of FO^p_k follows from (A2) as:

(A10) $$\frac{\partial FO^p_k}{\partial \beta^o_k} = \psi^p_k FO^p_k (1-FO^p_k) .$$

For the l-th hidden unit parameter β^h_l we have a derivation similar to (A9) until the last double summation:

$$(A11)\quad \Delta\beta_l^h = \eta \sum_p \sum_j \frac{\partial FO_j^p}{\partial \beta_l^h} (t_j^p - O_j^p) =$$

$$= \eta \sum_p \sum_j (t_j^p - O_j^p)\, FO_j^{p'} \frac{\partial(\sum_i w_{ij}^h H_i^p)}{\partial \beta_l^h} =$$

$$= \eta \sum_p \sum_j (t_j^p - O_j^p)\, FO_j^{p'} \sum_i w_{ij}^h \frac{\partial H_i^p}{\partial \beta_l^h} =$$

$$= \eta \sum_p \sum_j (t_j^p - O_j^p)\, FO_j^{p'}\, w_{lj}^h \frac{\partial H_l^p}{\partial \beta_l^h} =$$

$$= \eta \sum_p \sum_j (t_j^p - O_j^p)\, FO_j^{p'}\, w_{lj}^h \frac{\partial FH_l^p}{\partial \beta_l^h},$$

where the partial derivative of FH_l^p follows from (A1) as:

$$(A12)\quad \frac{\partial FH_l^p}{\partial \beta_l^h} = \phi_l^p FH_l^p (1 - FH_l^p) .$$

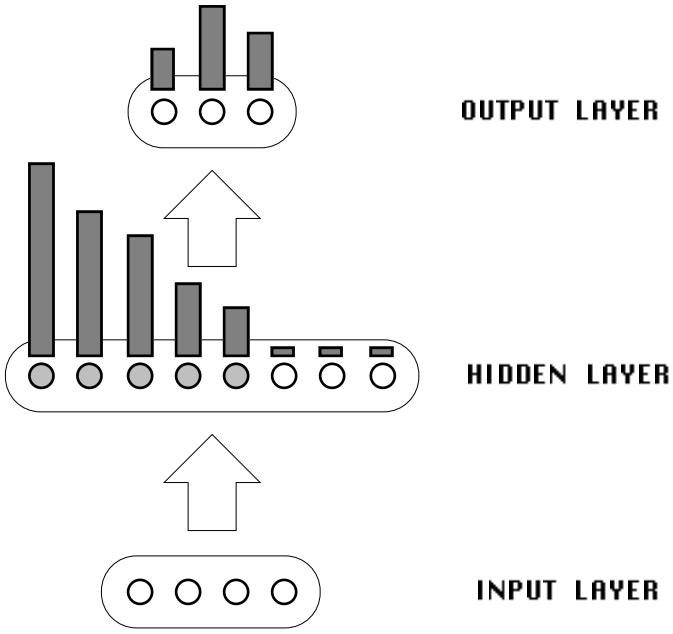

FIGURE 1.
Example of the proposed initial distribution of the steepness parameters in a three layer network. The height of the grey bars is proportional to the steepness values of the units. The values for a subset of the hidden units (shaded units) are distributed according to a decreasing function. The other units in the hidden layer do not have significant steepness values.

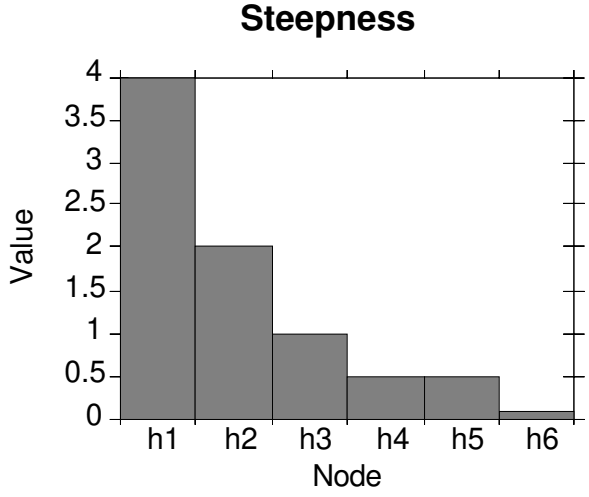

FIGURE 2.
The initial distribution SD2 for the steepness parameters of the hidden units in the four bits parity problem.

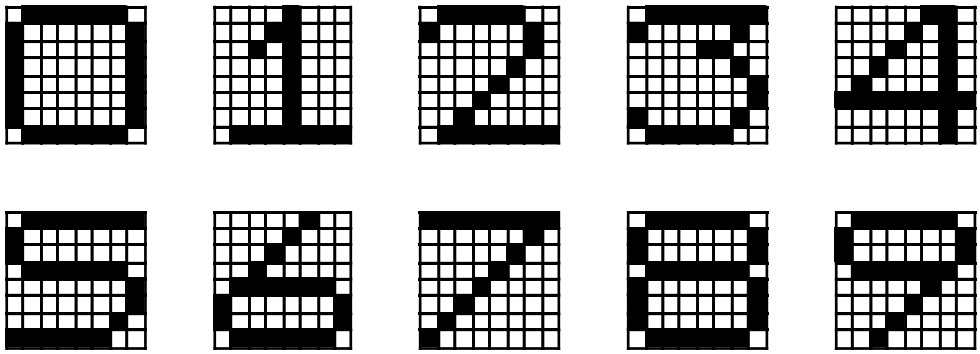

FIGURE 4.
The training set for the 8x8 dot numeric font learning.

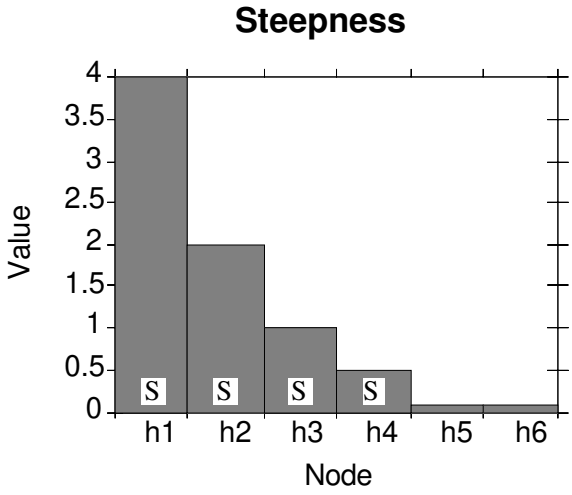

FIGURE 7.
Initial steepness distribution used for network optimization in the four bits parity problem. The hidden units h1-4 constitute the S set (units with initial relevant steepness values).

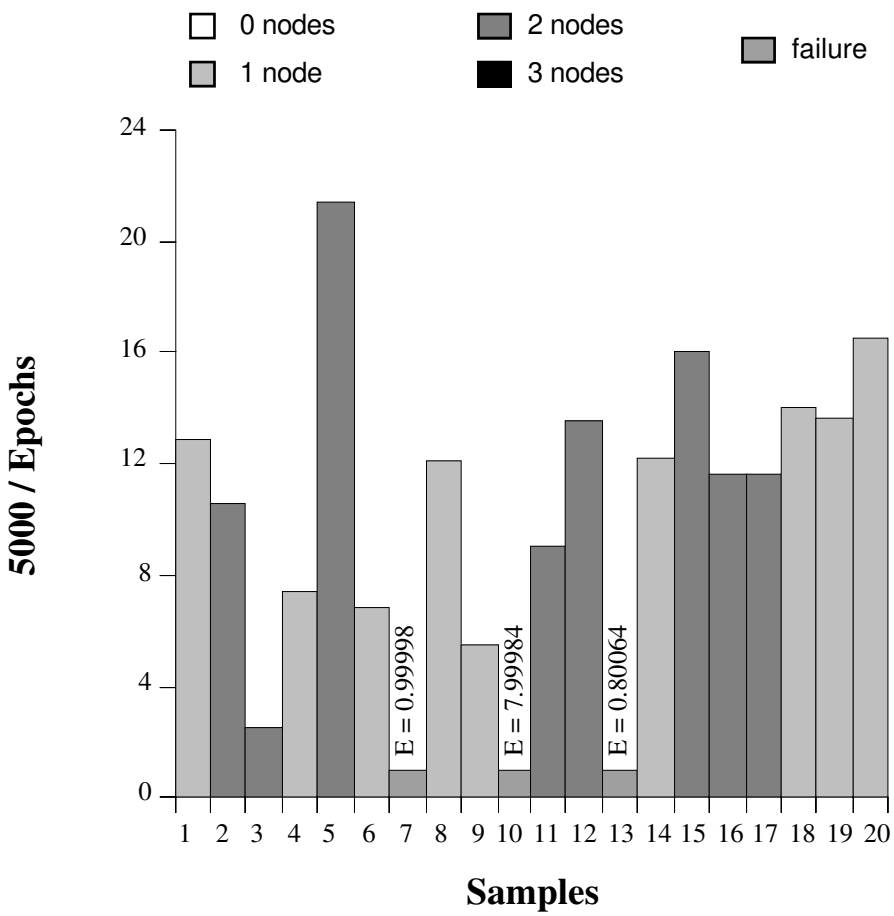

FIGURE 8.
Results of the optimization procedure over 20 different initial points in the weight space for the four bits parity problem with initial seepness distribution shown in Fig. 7. The network used was a 4-6-1. The patterns of the bars show how many units can been removed from the final solution and the height of the bars is proportional to the velocity of the learning (5000/number of epochs needed to converge). The total error (E) for simulations that failed to converge is reported. The setting of the learning parameters was $\mu = 0.9$, $\eta = \varepsilon = 0.3$ ($\eta / \varepsilon = 1$), and $\gamma = 0.001$.

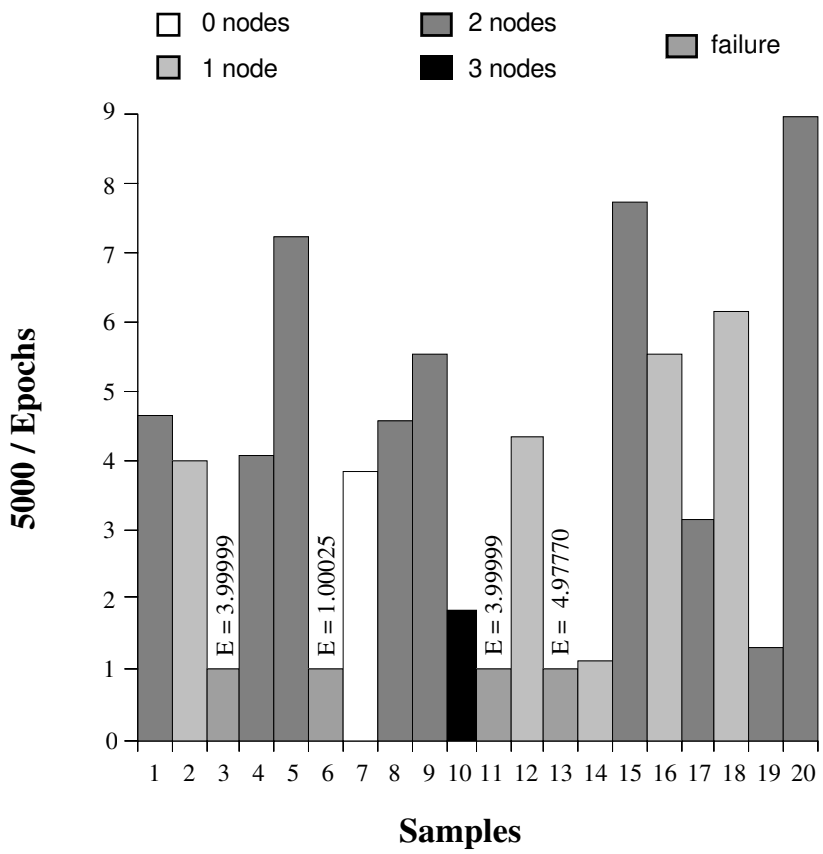

FIGURE 9.
Results of the optimization procedure with learning parameters $\mu = 0.9$, $\eta = 0.05$, $\varepsilon = 0.5$ ($\eta / \varepsilon = 0.1$), and $\gamma = 0.001$. Everything else as in Fig. 8. With this setting the learning was able to find a solution with only 3 hidden units (sample 10). Note also that the learning was slowed down by this setting.

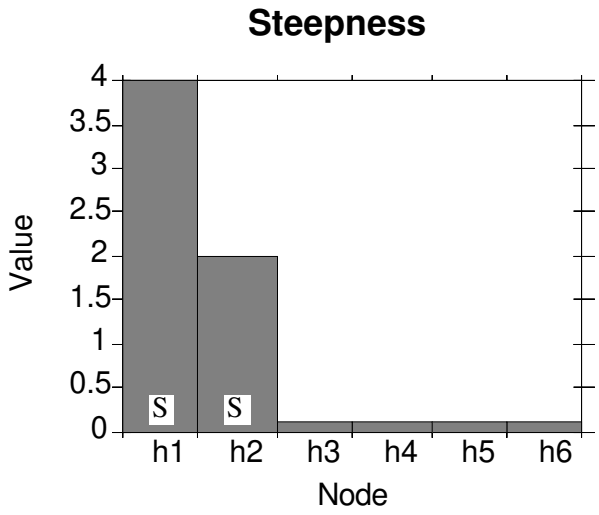

FIGURE 10.
Initial steepness distribution used to study the behavior of the learning procedure when the minimal number of hidden units needed to implement the four bits parity is underestimated. The hidden units h1-2 constitutes the S set (units with initial relevant steepness values).

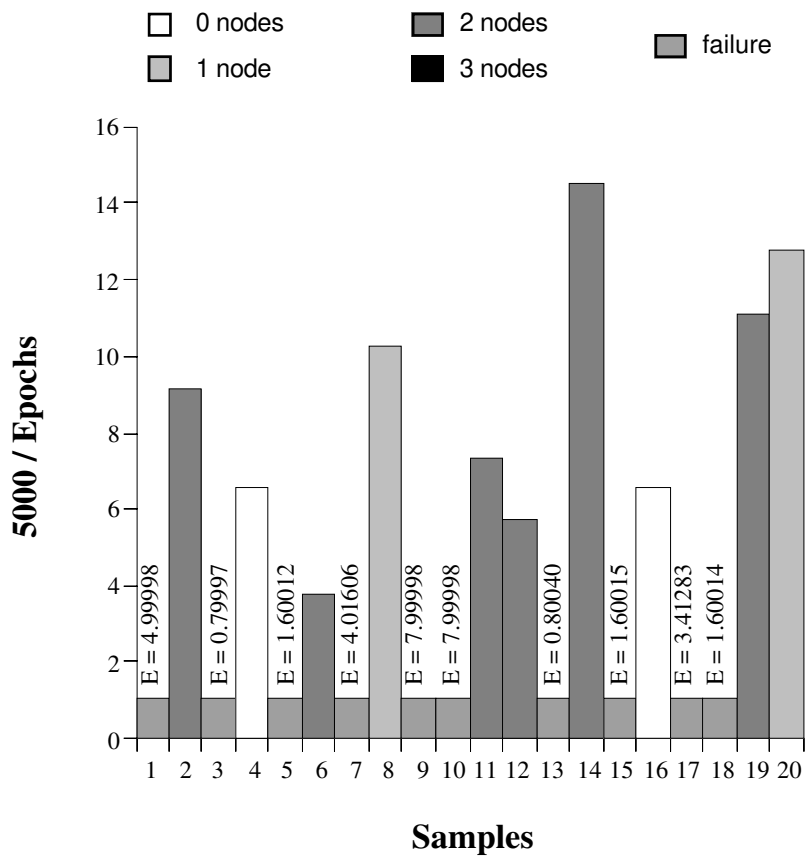

FIGURE 11.
Results of the optimization procedure with the initial steepness distribution shown in Fig. 10 and other settings as in Fig. 8. Several simulations were unable to converge in 5000 epochs.

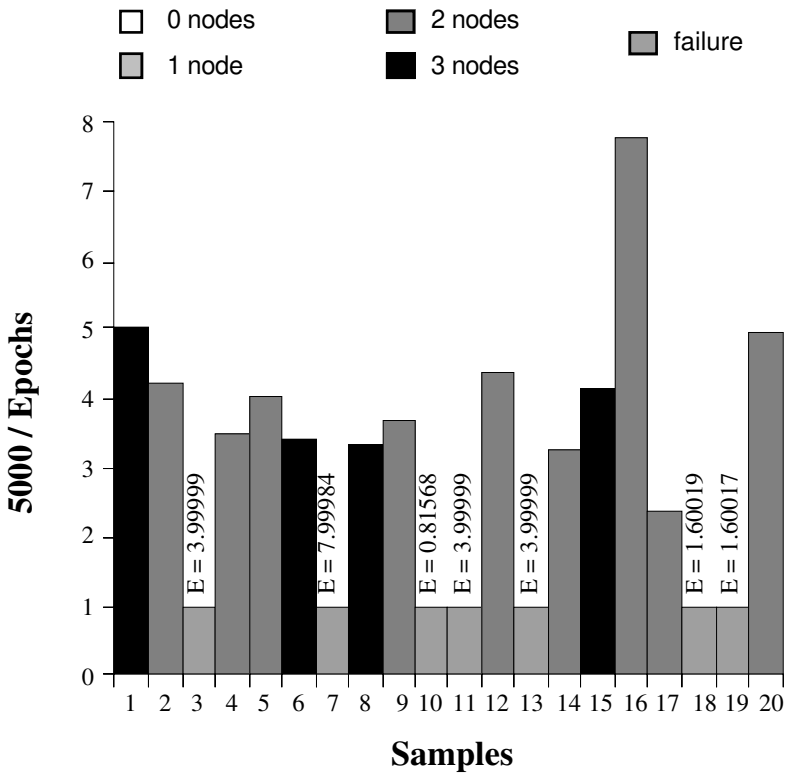

FIGURE 12.
Results of the optimization procedure with initial steepness distribution shown in Fig. 10 and other settings as in Fig. 9. Seven simulations were unable to converge in 5000 epochs, but all the others obtained good optimization performance. A solution was discovered with only three hidden units in four cases.

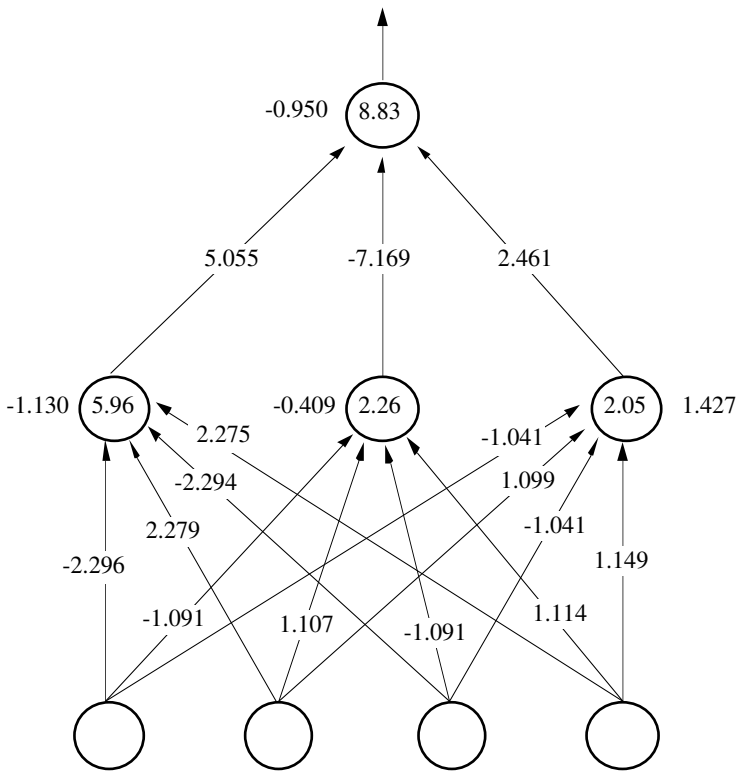

FIGURE 13.
The solution obtained for sample 6 in Fig. 12. The numbers in the circles refers to the steepness values of the units. Biases values are reported on the left or right of the units.

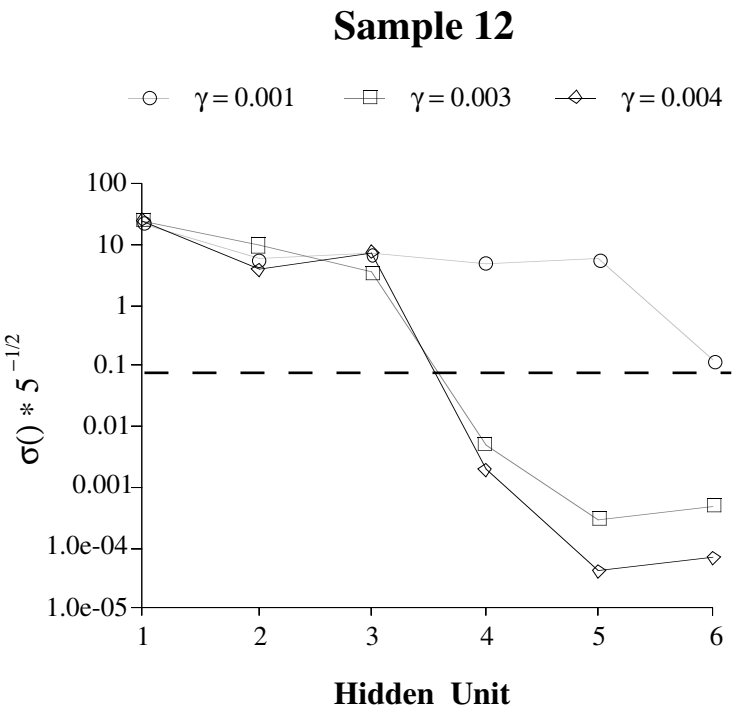

FIGURE 14.
Sensitivity curves for sample 12 in Fig. 9. The sensitivity of the last three units decreases when the decay rate γ is increased. Units with sensitivity value near or below the dashed line can be considered insensitive, and thus removable.

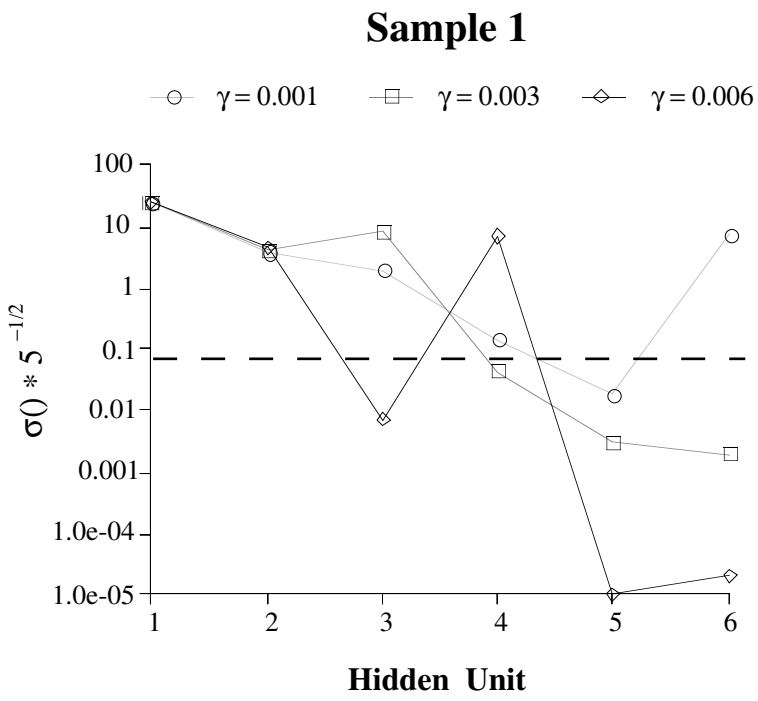

FIGURE 15.
Sensitivity curves for sample 1 in Fig. 9. The sensitivities of Units 3 and 4 show an irregular behavior (compared to Fig. 14) when the decay rate γ is increased. This shows that there is a complex relationship between the sensitivity and the learning parameters.