Neural Networks, Vol. 5, pp. 465–471, 1992
Printed in the USA. All rights reserved.

0893-6080/92 $5.00 + .00

ORIGINAL CONTRIBUTION

Improving the Convergence of the Back-Propagation Algorithm

A. VAN OOYEN
Netherlands Institute for Brain Research

AND

B. NIENHUIS
University of Amsterdam

(*Received* 26 *June* 1991; *revised and accepted* 17 *October* 1991)

Abstract—*We propose a modification to the back-propagation method. The modification consists of a simple change in the total error-of-performance function that is to be minimized by the algorithm. The modified algorithm is slightly simpler than the original. As a result, the convergence of the network is accelerated in two ways. During the learning process according to the original back-propagation method, the network goes through stages in which the improvement of the response is extremely slow. These periods of stagnation are much shorter or even absent in our modified method. Furthermore, the final approach to the desired response function, when the network is already nearly correct, is accelerated by an amount that can be predicted analytically. We compare the original and the modified method in simulations of a variety of functions.*

Keywords—Back-propagation, Gradient-descent, Three-layer networks, Error function, Convergence.

1. THE BACK-PROPAGATION METHOD

The back-propagation method of Rumelhart, Hinton, and Williams (1986a) is a learning procedure for multilayered, feedforward neural networks. By means of this procedure, the network can learn to map a set of inputs to a set of outputs. The mapping is specified by giving the desired activation state of the output units (the target state vector) for each presented state of the input units (the input state vector). Learning is carried out by iteratively adjusting the coupling strengths in the network so as to minimize the differences between the actual output state vector of the network and the target state vector. The network is initialized with small random coupling strengths. During the learning process, an input vector is presented to the network and propagated forward to determine the output signal. The output vector is then compared with the target vector resulting in an error signal, which is back-propagated through the network in order to adjust the coupling strengths. This learning process is repeated until the network responds for each input vector with an output vector that is sufficiently close to the desired one. The back-propagation algorithm will now be described in more detail reproducing the formulas from Rumelhart et al. (1986a, 1986b).

The general formula for the activation A of each unit in the network (except for the input units whose activation is clamped by the input vector) is given by:

$$A_j(w, h, y) = \frac{1}{1 + \exp\{-(\sum_{i=1}^{N}(w_{ji}y_i) + h_j)\}}, \quad (1)$$

where w_{ji} is the strength of the coupling between unit j (for which the activation is calculated) and unit i in the next lower layer, N is the total number of units in that layer, y_i is the activation of unit i, and h_j is the threshold or bias for unit j. This bias can be conceived of as a coupling to a unit with full activation, and is in practice treated just like w. Here we consider a three-layered network consisting of a layer of input units (represented by x), a layer of hidden units (y), and a layer of output units (z). The activation of a hidden unit,

$$y_j = A_j(v, h_j, x), \quad (2)$$

where v represents the strength of the couplings between

Requests for reprints should be sent to A. van Ooyen, Netherlands Institute for Brain Research, Meibergdreef 33, 1105 AZ Amsterdam, The Netherlands.

layer x and y, and h_j is the bias of hidden unit j. The activation of an output unit,

$$z_j = A_j(w, o_j, y), \tag{3}$$

where w represents the strength of the couplings between layer y and z, and o_j is the bias of output unit j. The total error of the performance of the network, E, is defined as:

$$E = \frac{1}{2}\sum_{c=1}^{N_c}\sum_{j=1}^{N_z}(z_{j,c} - t_{j,c})^2, \tag{4}$$

where c runs over all cases (input vectors with their corresponding target vectors), N_c is the total number of cases, z_j is the actual value (activation) of output unit j, given the input vector, N_z is the total number of output units, and t_j is the target value of unit j. Hereafter we will suppress the case index c, where clarity does not require it. To minimize E, each coupling strength is updated by an amount proportional to the partial derivative of E with respect to that coupling (accumulated over all cases). The computation of the partial derivatives will be given in some detail because the intermediate results are used also elsewhere in this paper. The partial derivative of E with respect to w_{ji},

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}}, \tag{5}$$

where

$$\frac{\partial E}{\partial z_j} = z_j - t_j, \tag{6}$$

and

$$\frac{\partial z_j}{\partial w_{ji}} = z_j(1 - z_j)y_i. \tag{7}$$

The partial derivative of E with respect to v_{ik},

$$\frac{\partial E}{\partial v_{ik}} = \sum_{j=1}^{N_z}\frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial y_i}\frac{\partial y_i}{\partial v_{ik}}, \tag{8}$$

where

$$\frac{\partial z_j}{\partial y_i} = z_j(1 - z_j)w_{ji}, \tag{9}$$

and

$$\frac{\partial y_i}{\partial v_{ik}} = y_i(1 - y_i)x_k. \tag{10}$$

The factor $(z_j - t_j)z_j(1 - z_j)$ occurs in both $\partial E/\partial w_{ji}$ and $\partial E/\partial v_{ik}$; $(z_j - t_j)z_j(1 - z_j)$ is, as it were, propagated backward from the layer of output units to the layer of hidden units. As it contains only z_j and t_j, it is defined as the error signal of output unit j.

The coupling strengths w_{ji} are updated according to the following rule (Rumelhart et al., 1986a; Vogl, Mangis, Rigler, Zink, & Alkon, 1988):

$$\Delta w_{ji}(s + 1) = -\eta\sum_{c=1}^{N_c}\left(\frac{\partial E}{\partial w_{ji}}\right) + \alpha\Delta w_{ji}(s), \tag{11}$$

where s represents the sweep number (i.e., the number of times the network has been through the whole set of cases, at which time the coupling strengths are updated), c runs over cases, N_c is the total number of cases, η is the (constant) learning rate, α is the relative contribution of the previous change of the coupling strength (α is the so called momentum factor). By analogy, the coupling strengths v_{ik} are updated as:

$$\Delta v_{ik}(s + 1) = -\eta\sum_{c=1}^{N_c}\left(\frac{\partial E}{\partial v_{ik}}\right) + \alpha\Delta v_{ik}(s). \tag{12}$$

The back-propagation algorithm amounts to performing gradient descent on a hyper surface in coupling strength space, where at any point in that space the error of performance (4) is the height of the surface. The method is not guaranteed to find a global minimum of E since gradient descent may get stuck in (poor) local minima, where it will stay indefinitely.

In practice, back-propagation has proved to be a suitable algorithm in establishing a set of coupling strengths that enables the network to perform certain input-output mappings (e.g., Rumelhart et al., 1986b; Sejnowski & Rosenberg, 1987; Zipser & Andersen, 1988). The convergence, however, tends to be extremely slow. Several acceleration techniques have been proposed [e.g., dynamically modifying the learning parameters (Fahlman, 1987; Jacobs, 1988; Tollenaere, 1990; Vogl et al., 1988), and rescaling the partial derivatives in the consecutive layers (Rigler, Irvine, & Vogl, 1991)].

2. THE MODIFIED BACK-PROPAGATION METHOD

The back-propagation algorithm as described above encounters the following difficulty. When the actual value z_j approaches either extreme value, the factor $z_j(1 - z_j)$ in eqn (7) makes the error signal very small. This implies that an output unit can be maximally wrong without producing a strong error signal with which the coupling strengths could be significantly adjusted. This retards the search for a minimum in the error. For instance, this occurs when some of the output units are pushed towards the wrong extreme value by competition in the network, thereby not increasing their error signal but instead decreasing it.

We note that this delay of the convergence is caused by the derivative of the activation function. Unfortunately, any saturating response function is bound to have this property: near the saturation points the derivative vanishes. We will show, however, that a slightly modified error function of the back-propagation algorithm resolves this shortcoming and indeed greatly accelerates the convergence to a solution. This applies not only to the initial approach of the desired values, but also to the final convergence process when the response is already near the target vector.

The total error function that is to be minimized by gradient descent, is chosen more or less ad hoc. Other functions that may be more appropriate with respect to minimization, could be applied as well. Instead of minimizing the squares of the differences between the actual and target values summed over the output units and all cases, we propose the following error function to be minimized

$$E = -\sum_{c=1}^{N_c} \sum_{j=1}^{N_z} [t_j \ln z_j + (1 - t_j) \ln(1 - z_j)], \quad (13)$$

where c runs over cases, N_c is the total number of cases, N_z is the total number of output units, z_j is the actual value (between 0 and 1) of output unit j, and t_j its target value. In the modified algorithm, the partial derivative of E with respect to z_j,

$$\frac{\partial E}{\partial z_j} = \frac{1 - t_j}{1 - z_j} - \frac{t_j}{z_j} = \frac{z_j - t_j}{z_j(1 - z_j)}. \quad (14)$$

As $\partial z_j / \partial w_{ji}$ is not altered in the modified algorithm, the partial derivative of E with respect to w_{ji} is now

$$\frac{\partial E}{\partial w_{ji}} = (z_j - t_j) y_i \quad (15)$$

instead of $(z_j - t_j) z_j (1 - z_j) y_i$ as in the original algorithm. In the case that the target values are 0 or 1, formula (13) is equivalent to the one proposed independently by Hinton (1987) and which we know through a citation by Golden (1988).

Thus, the error signal, propagating back from each output unit, is now directly proportional to the difference between target value and actual value. Formula (15) lacks the factor $z_j(1 - z_j)$, so "true" error is measured. Only this minimal modification needs actually be implemented in the original algorithm to change it into our modified method.

3. COMPARISON OF THE TWO METHODS

The effect of the substitution of the error function can in part be predicted directly from the algorithm. We introduce the following distance between the current output signal and the target values

$$\|z - t\| \equiv \left(\sum_{c=1}^{N_c} \sum_{j=1}^{N_z} (z_{j,c} - t_{j,c})^2 \right)^{1/2}. \quad (16)$$

The variation in this distance with time can be simply computed, as yet without specification of the error function

$$\Delta\|z - t\| = \sum_{c=1}^{N_c} \sum_{i=1}^{N_y} \sum_{j=1}^{N_z} \frac{\partial\|z - t\|}{\partial z_j} \times \left(\frac{\partial z_j}{\partial w_{ji}} \Delta w_{ji} + \sum_{k=1}^{N_x} \frac{\partial z_j}{\partial v_{ik}} \Delta v_{ik} \right). \quad (17)$$

With eqs (11 and 12) this can be rewritten

$$\Delta\|z - t\| = -(\alpha + \eta) \sum_{c=1}^{N_c} \sum_{i=1}^{N_y} \sum_{j=1}^{N_z} \frac{\partial\|z - t\|}{\partial z_j} \frac{\partial E}{\partial z_j} \times \left[\left(\frac{\partial z_j}{\partial w_{ji}} \right)^2 + \sum_{k=1}^{N_x} \left(\frac{\partial z_j}{\partial y_i} \frac{\partial y_i}{\partial v_{ik}} \right)^2 \right]. \quad (18)$$

Note that both terms of the right hand side contain from the derivatives of z_j two powers of $z_j(1 - z_j)$. These factors suppress the improvement of the network whenever any of the z_j in any of the cases approaches either 0 or 1. When, however, we choose the error function (13), the derivative $\partial E / \partial z_j$ improves the situation by taking out one factor $z_j(1 - z_j)$. Equation (18) already suggests that the algorithm with the original error function (4) is susceptible to very slow change whenever the values of z_j are all near 0 or 1. This is problematic when some of the z_j values are not near their target value. In the modified algorithm this problem is largely overcome, as already seen here by simply counting powers, and as shown below in simulations.

A more quantitative analysis is possible when we study the final convergence. In the tail of a successful learning process we may assume that all $(z_j - t_j)$ are small. As $\|z - t\|$ approaches zero also its change vanishes, in fact as a power of $\|z - t\|$

$$\Delta\|z - t\| \sim \|z - t\|^q, \quad (19)$$

where for binary problems $q = 3$ in the original and $q = 2$ in the modified algorithm. In both cases the constant of proportionality will depend on many details of the network. Since this change can be read as a derivative with respect to time, or sweep number s, we find that

$$\|z - t\| \sim s^{-p}, \quad (20)$$

where $p = \frac{1}{2}$ for the original and $p = 1$ for the modified algorithm. Thus, we have demonstrated that our modification improves the final convergence. For the initial approach we have shown that the modified network is less apt to get stuck in states in which some of the output cells are approaching the wrong value.

3.1. Simulation Results

To compare both methods as to how many presentations of the set of input vectors are required to achieve a solution, a wide range of binary problems were examined. At present, it is not clear how one could, in advance, determine the complexity of a problem for the back-propagation algorithm. We therefore choose problems of which we think are roughly of different complexity. A network with only one layer of hidden units was used. Each unit of the hidden and output layer is connected with every unit in the next lower layer; there are no direct couplings between the output layer and the input units. As all gradient descent procedures, the back-propagation procedure is sensitive to different starting points (here: the initial set of coupling strengths in the network). Therefore, we experimented with runs starting from different random initializations of the network. The same initializations were used for both the modified method and the original method. The initial coupling strengths were drawn at random from an uniform distribution between −0.3 and 0.3 (unless otherwise stated). No fair comparison is possible if the learning rate (η) and the momentum factor (α) are kept the same for both methods, since the actual learning rate in the modified method is always higher compared with that in the original method. This as a result of dropping the factor $z_j(1 - z_j)$, which has a maximum value of 0.25. In order to compare both methods fairly, we tried to establish the optimum values of α and η for each method and problem separately (Tollenaere, 1990). Using runs starting from different initializations of the network, the learning parameters were optimized with respect to the average number of sweeps that is required to reach the stage of the network in which the maximal error of the output units, measured as $|z_j - t_j|$, is less than 0.5 (and remains less than 0.5). No attempts were made to find the optimal values of the learning parameters for each run (initialization) separately. At the time $\max(|z_j - t_j|, 1 \leq j \leq N_z) < 0.5$, all the output units are on the 'right' side. The convergence to this stage of the network is defined as the initial approach to the solution. The convergence to the target values from the direct vicinity of the target values is defined as the final convergence. For each problem examined, the number of sweeps that are required for the initial approach will be given for both methods. The figures, in which the performance error, defined as $\Sigma_c \Sigma_j (z_{j,c} - t_{j,c})^2$ (see eqn (4)), is plotted against sweep number, show the convergence behavior of the original and modified method for different problems.

3.1.1. *The XOR problem.* The exclusive-or (XOR) problem was presented to a network consisting of a layer of two input units, a layer of two hidden units, and one output unit. Ten different initializations of the network were generated using an uniform distribution between −1.3 and 1.3. The original back-propagation method using a $\eta = 2.6$ and an $\alpha = 0.9$ required on average 64 (standard deviation (σ) = 29) sweeps through the four input vectors while the modified method using an an $\eta = 0.7$ and an $\alpha = 0.8$ required on average 61 sweeps ($\sigma = 24$). Thus, with this relative simple problem both methods achieve a solution (initial approach) after about the same number of sweeps. The final convergence to the solution is, of course, much faster with the modified method.

3.1.2. *The encoding problem.* This is a problem in which a set of orthogonal input vectors are mapped to a set of orthogonal output vectors through a small set of hidden units (Ackley, Hinton, & Sejnowski, 1985). The network used here consists of eight input units, three hidden units, and eight output units, and is to learn the identity mapping. In each of the eight input vectors precisely one of the input units has activation one and the remaining units have activation zero. The corresponding target output vector has exactly the same activation pattern. The network is thus to learn an encoding of an eight bit input vector into a three bit pattern and its decoding into an eight bit output vector. Ten different initializations of the network were generated. Initial approach was reached after on average 153 sweeps ($\sigma = 45$) with the original method using a $\eta = 1.9$ and an $\alpha = 0.6$, while on average 91 sweeps ($\sigma = 27$) were required with the modified method using a $\eta = 0.7$ and an $\alpha = 0.3$. With the original method, there is a period at the beginning of each run in which the total error changes only slowly (appearing as a plateau in figure 1). With the modified method, this period is shorter, as a result of which the initial approach is faster.

3.1.3. *The F-function.* For this function, we consider a k-dimensional matrix X of size m^k. The binary $\{0,1\}$ entries of X are denoted as $x(i_1, i_2, i_3, \ldots, i_k)$. Now, on matrix X the F-function (Wegener, 1987) is defined as:

$$F_{k,n}(x) = 1 \text{ iff } \exists i_1 \, \forall i_2 \, \exists i_3 \ldots Qi_k : x_{i_1,i_2,i_3,\ldots\, i_k} = 1, \quad (21)$$

where $Q = \forall$ for k is even and $Q = \exists$ for k is odd. This function can easily be implemented in a multilayered, feedforward network that consists of alternating layers of AND and OR cells, and one layer of input cells whose activation represent the entries of matrix X. For example, to construct a network that is to perform the $F_{3,8}$ function, one would need a 4-layered network consisting of eight input cells, a layer of four OR cells, a layer of two AND cells, and, finally, one OR cell, whose output determines whether or not $F_{3,8} = 1$. Here, the problem of learning the $F_{3,8}$ function was posed to a 3-layered network so that the network cannot simply develop AND and OR units. We used a network consisting of eight input units, five hidden units, and one output unit. At each sweep, the whole set of 2^8 input vectors was presented. Five different initializations of the network were generated. The modified method using a $\eta = 0.02$ and an $\alpha = 0.7$ turned out to be two to three times as fast as the original method using a $\eta = 0.03$ and an $\alpha = 0.8$. The initial approach required on average 99 sweeps ($\sigma = 41$) with the modified method and on average 212 sweeps ($\sigma = 70$) with the original method. Again, the acceleration is mainly due to the fact that the 'plateaus in error,' in which the total error changes only slowly, are shorter (see figure 2).

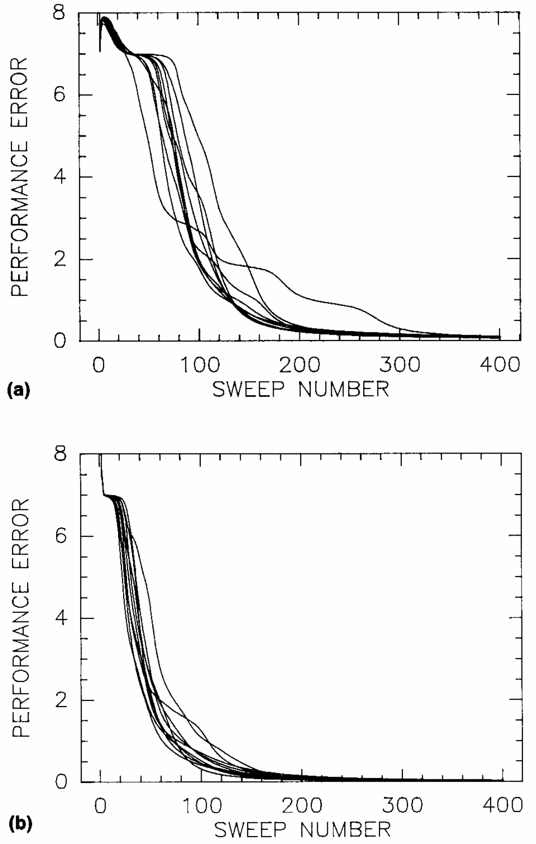

FIGURE 1. Convergence behavior for the encoding problem using the original back-propagation method (a), and using the modified method (b). Runs starting from ten different initializations of the network are shown.

3.1.4. *The counting problem.* For the counting problem the network is to count the number of input units that are on, and to generate a unary representation of this number in the output layer. The specific problem used here is counting the number of 1's in an input vector of length 4 using a network of four input units, four hidden units, and five output units. In each of the 16 target vectors, precisely one desired state of the output units has activation one while the remaining units have activation zero. Only the 'first' output unit should be active if there are no 1's occurring in the input vector, only the second if there is one 1 in the input vector, only the third if there are two 1's, etc. Using a $\eta = 0.6$ and an $\alpha = 0.7$, initial approach with the original method was achieved within 1,000 sweeps (through the 16 input vectors) in only 4 out of 10 different runs. Of these, two runs required more than 900 sweeps (913 and 968). The other two runs required 324 and 217 sweeps. Of the other six runs, one run required about 3,500 sweeps to achieve initial approach. For the rest, initial approach was not achieved after more than 10,000 sweeps. With the modified method, using a $\eta = 0.2$ and an $\alpha = 0.6$, initial approach was achieved after less than 230 sweeps for as many as 8 runs (average number of sweeps required by these eight runs is 188, $\sigma = 19$). Only two runs required more sweeps (752 and 1048). Of all the problems we studied so far, the counting problem most convincingly demonstrates the improved performance obtained by using the modified method. Figure 3 shows that, with the original method, the network encounters 'plateaus,' in which the total error changes extremely slowly. With the modified method, these plateaus are much shorter or do not exist at all.

4. CONCLUSIONS AND DISCUSSION

In this paper, we have demonstrated that a simple modification to the back-propagation algorithm accelerates its convergence. The modification amounts to changing the total error function that is to be minimized by the algorithm. As a result the error signal produced by an output unit is now directly proportional to the

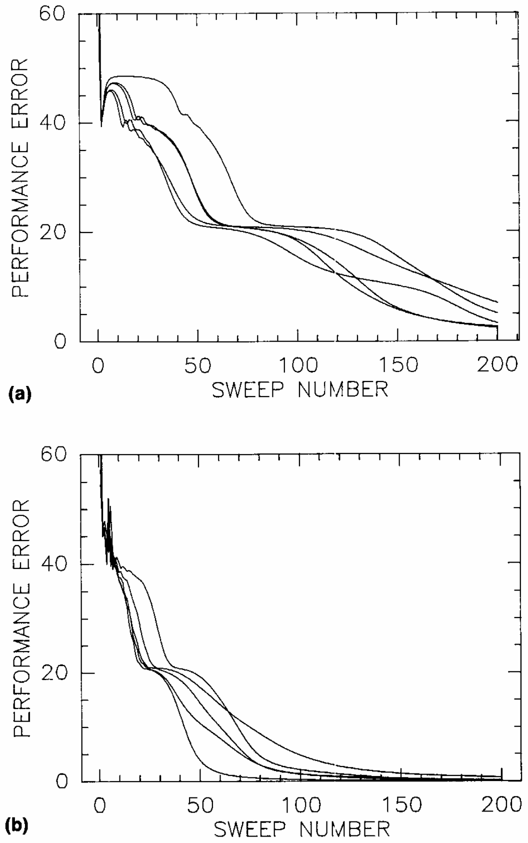

FIGURE 2. Convergence behavior for the *F*-function problem using the original back-propagation method (a), and using the modified method (b). Runs starting from five different initializations of the network are shown.

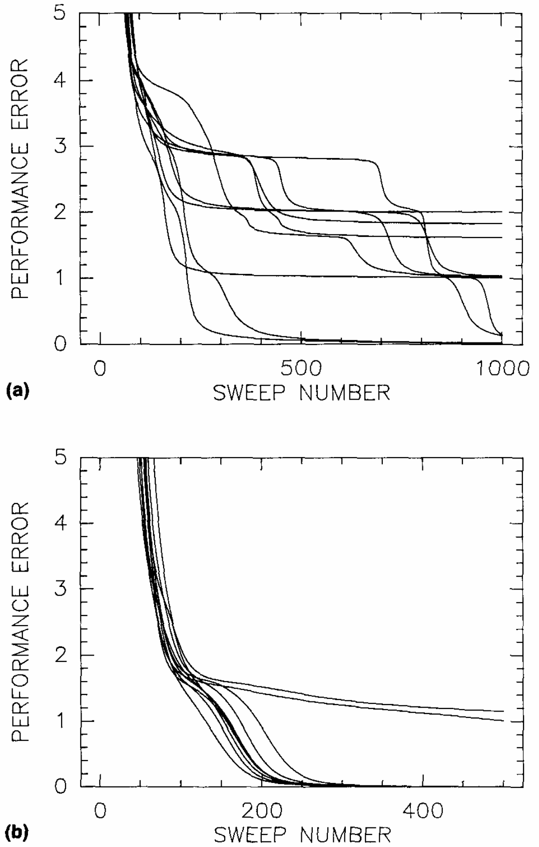

FIGURE 3. Convergence behavior (shown from sweep number is about 100) for the counting problem using the original back-propagation method (a), and using the modified method (b). Runs starting from ten different initializations of the network are shown.

difference between the actual activation of the unit and its target value. A strong error signal is produced also when an output unit approaches the value 0 or 1, provided it is not the target value. With the original algorithm, only a very small error is produced when an output unit approaches a wrong extreme value, which slows down the convergence.

With the modified method, the initial approach as well as the final convergence are faster. The acceleration of the final convergence was shown analytically and by simulation on binary problems. In all the problems we studied so far, the initial approach is faster or is at least the same. The performance is most improved in difficult problems. With the original method, there are periods of stagnation, during which the performance of the network improves only slowly. They were observed in all the problems but most clearly in the counting problem. With the modified method, the initial approach is faster mainly because these periods of stagnation are much shorter in time or do not exist at all.

Closely studying the behavior of the network revealed that by competition some output units can be pushed towards the wrong extreme value while other units are reaching there target value and thus decreasing the total error. With the original method, there is only a small error signal for an output unit being near the wrong extreme value. As a result, the network can continue to move towards a state whose response is correct for all but a few cases. Because the signal that is propagated backward is very small, the performance of the network hardly improves: a plateau in error arises. As soon as the network is responding correctly for most cases, the small error produced by units that are near the wrong extreme value for the few remaining cases, can be noticed and dealt with. (Note that Δw is a sum over all changes in coupling strength that are calculated for each case separately). The network now adjusts the coupling strengths, albeit slowly because initially the error signal is very small. During this process, the total error is hardly changing until a certain threshold is reached after which the network changes exponentially. When this state is reached depends on how far the output units are pushed to the wrong extreme value. After escaping from one plateau, the network can possibly encounter a new one, as was observed in the counting problem. This, of course, further delays convergence. With the modified method on the other hand, plateaus in error are not likely to occur because a strong error signal is propagated as soon as the output units are approaching the wrong extreme value. One might say that with the modified algorithm competition among cases and among units is more balanced.

We have not carried out simulation studies to compare this method of improvement with other acceleration schemes. Adaptive schemes, which adjust the learning parameters to the local error landscape in order to avoid convergence delays, approach the problem from the symptomatic side. Our method takes away the cause of a part of those convergence delays.

REFERENCES

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, **19**, 147–169.

Fahlman, S. (1987). *An empirical study of learning in back-propagation networks* (CMU-CS-88-162). Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Golden, R. M. (1988). A unified framework for connectionist systems. *Biological Cybernetics*, **59**, 109–120.

Hinton, G. E. (1987). *Connectionist learning procedures* (CMU-CS-87-115). Department of Computer Science Technical Report, Carnegie-Mellon University, Pittsburgh, PA.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaption. *Neural Networks*, **1**, 295–307.

Rigler, A. K., Irvine, J. M., & Vogl, T. P. (1991). Rescaling of variables in back propagation learning. *Neural Networks*, **4**, 225–229.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986a). Learning representations by back-propagating errors. *Nature*, **323**, 533–536.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986b). Learning internal representations by error back propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Vol 1: Foundations* (pp. 318–362). Cambridge, MA: MIT Press.

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems,* **1,** 145–168.

Tolleneare, T. (1990). SuperSAB: Fast adaptive back propagation with good scaling properties. *Neural Networks,* **3,** 561–573.

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.

Wegener, I. (1987). *The complexity of Boolean functions.* Stuttgart: Wiley/Teubner.

Zipser, D., & Andersen, R. A. (1988). A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature,* **331,** 679–684.

NOMENCLATURE

x_j	Activation of unit j of the input layer
y_j	Activation of unit j of the hidden layer
z_j	Activation of unit j of the output layer
$t_{j,c}$	Desired output value of unit j for input case c
A_j	Activation function of unit j as function of its coupling to cells in previous layers and of their activation
h_j	Threshold or bias of unit j of the hidden layer
o_j	Threshold or bias of unit j of the output layer
w_{ji}	Coupling strength from unit i in the hidden layer to unit j in the output layer
v_{ji}	Coupling strength from unit i in the input layer to unit j in the hidden layer
E	The error function (i.e., the accumulation of a measure of all the deviations from the desired output, summed over the cases and output units)
N_c	Number of cases, or different input configurations for which a specific output activation is desired
N_x	Number of units in the input layer
N_y	Number of units in the hidden layer
N_z	Number of units in the output layer
η	Learning rate
α	Momentum factor