Complex Systems **3** (1989) 331–342

Accelerated Backpropagation Learning: Two Optimization Methods

Roberto Battiti*
Caltech Concurrent Computation Program,
206-49 California Institute of Technology, Pasadena, CA 91125, USA

Abstract. Two methods for increasing performance of the backpropagation learning algorithm are presented and their results are compared with those obtained by optimizing parameters in the standard method. The first method requires adaptation of a scalar learning rate in order to decrease the energy value along the gradient direction in a close-to-optimal way. The second is derived from the conjugate gradient method with inexact linear searches. The strict locality requirement is relaxed but parallelism of computation is maintained, allowing efficient use of concurrent computation. For medium-size problems, typical speedups of one order of magnitude are obtained.

1. Introduction

Multilayer feedforward "neural" networks have been shown to be a useful tool in diverse areas, such as pattern classification, multivariable functional approximation, and forecasting over time [2,3,6,8,9,11]. In the "backpropagation" learning procedure, a network with a fixed structure is programmed using gradient descent in the space of the weights, where the *energy function* to be minimized is defined as the sum of squared errors [11].

In a given iteration n, the *search direction* $\mathbf{d}_n$ is defined as the negative gradient of the energy, while the step $\Delta\mathbf{w}_n$ along this direction is taken to be proportional to $\mathbf{d}_n$ with a fixed constant ϵ ("learning rate"), as follows:

$$\mathbf{d}_n = -\nabla E(\mathbf{w}_n) \tag{1.1}$$

$$\Delta\mathbf{w}_n = \epsilon\, \mathbf{d}_n \tag{1.2}$$

The learning rate is usually chosen by the user to be "as large as possible without leading to oscillations" [11].

It is well known from the optimization literature that pure gradient descent methods are usually very inefficient. For example, if the steepest descent method is applied to a quadratic function $F(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + 1/2\ \mathbf{x}^T G\mathbf{x}$

*Electronic mail address: roberto@hamlet.caltech.edu.

using an *exact line search* to determine the step length, it can be shown that

$$F(x_{n+1}) - F(x^*) \approx \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}\right)^2 (F(x_n) - F(x^*)) \tag{1.3}$$

where x^* is the optimal point and λ_{max} and λ_{min} are the largest and smallest eigenvalues of G. This means that the asymptotic error reduction constant can be *arbitrarily close to unity* [5]. A case in which this happens is when "the search space contains long ravines that are characterized by sharp curvature across the ravine and a gently sloping floor" [11]. The situation can be ameliorated in part modifying the search direction with the introduction of a *momentum* term (and a parameter α), leading to the following rule:

$$\mathbf{d}_n = -\nabla E(\mathbf{w}_n) + \left(\frac{\alpha}{\epsilon}\right) \Delta \mathbf{w}_{n-1} \tag{1.4}$$

$$\Delta \mathbf{w}_n = \epsilon \, \mathbf{d}_n \tag{1.5}$$

Recently, an overview of heuristics employed to accelerate backpropagation has been presented in [10], where it is suggested that each weight should be given its own learning rate, changing over time during the computation.

Unfortunately, there are *no general prescriptions* for the selection of the parameters defining the optimization strategy (like ϵ or α). It is usually left to the user to find a good or optimal combination of these parameters that leads to avoidance of local minima and fast convergence times. This is certainly interesting from the point of view of theoretical research ([15] is a good example), but leads in general to a waste of time and computational resources during this *meta-optimization* phase (optimization of the behavior of the optimization method).

The objection to using standard optimization techniques is usually that they require some sort of *global* computation. *Locality* is a concept that depends on the mapping between a given algorithm and the processors (e.g., VLSI hardware, biological neurons) responsible for the computation. In this sense, backpropagation is *local* if different processors are assigned to the different weights and "neurons" and if the chain rule for partial derivatives is used in calculating the gradient, "backpropagating" the errors through the network. A concept related but different from locality is that of *parallelism*, where a given computation can be done by more computational units working concurrently on different partial tasks (with a speedup in the time required to complete it). Despite the fact that networks performing local computation are usually easier to implement in parallel architectures, it is nonetheless true that parallelism of computation can be obtained also in certain cases where a global information exchange is required (see [4] for many examples in both areas).

The focus of this work has been on transferring some *meta-optimization* techniques usually left to the user to the learning algorithm itself. Since this involves measuring optimization performance and correcting some parameters while the optimization algorithm is running, some *global* information is

required (typically in the form of scalar products of quantities distributed over the network).

In all cases, the "standard" backpropagation algorithm is used to find the values of the energy and the negative gradient for a given configuration. The differences are in the definition of the *search direction* and/or in the selection of a *step size* along the selected direction.

In the first method proposed, the search direction remains equal to the negative gradient but the (scalar) step size is adapted during the computation. This strategy has been suggested in [7,16] and is here summarized for convenience before using it in the test problems. In the second, both the search direction and the step size are changed in a suboptimal but apparently very efficient way.

In both cases, the network is updated only after the entire set of patterns to be learned has been presented to it.

2. First method: the "bold driver" network

This method requires only a limited change to standard backpropagation. In general, the number of steps to convergence for a steepest descent method is a decreasing function of the learning rate up to a given point, where oscillations in the weights are introduced, the energy function does not decrease steadily, and good local minima are missed. Performance degradation in this case is usually rapid and unpredictable.

The proposed solution is to start with a given learning rate and to monitor the value of the energy function $E(\mathbf{w}_n)$ after each change in the weights. If E decreases, the learning rate is then increased by a factor ρ. Vice versa, if E increases, this is taken as an indication that the step made was too long and the learning rate is decreased by a factor σ, the last change is cancelled, and a new trial is done. The process of reduction is repeated until a step that decreases the energy value is found (this will be found if the learning rate is allowed to tend to zero, given that the search direction is that of the negative gradient).

Heuristically, ρ has to be close to unity (say $\rho \approx 1.1$) in order to avoid frequent "accidents," because the computation done in the last backpropagation step is wasted in these cases. Regarding the parameter σ a choice of $\sigma \approx 0.5$ can be justified with the reason that if the local "ravine" in the search space is symmetric on both sides this will bring the configuration of the weights close to the bottom of the valley.

The exponential increase in the learning rate ($\epsilon = \epsilon_0 \rho^n$) is preferred to a linear one because it *will* typically cause an "accident" after a limited number of steps, assuming that the proper learning rate for a given terrain increases less rapidly. Such accidents are productive because after them, the learning rate is reset to a value appropriate to the local energy surface configuration.

An example for the size of the learning rate as a function of the iteration number is given in figure 1.

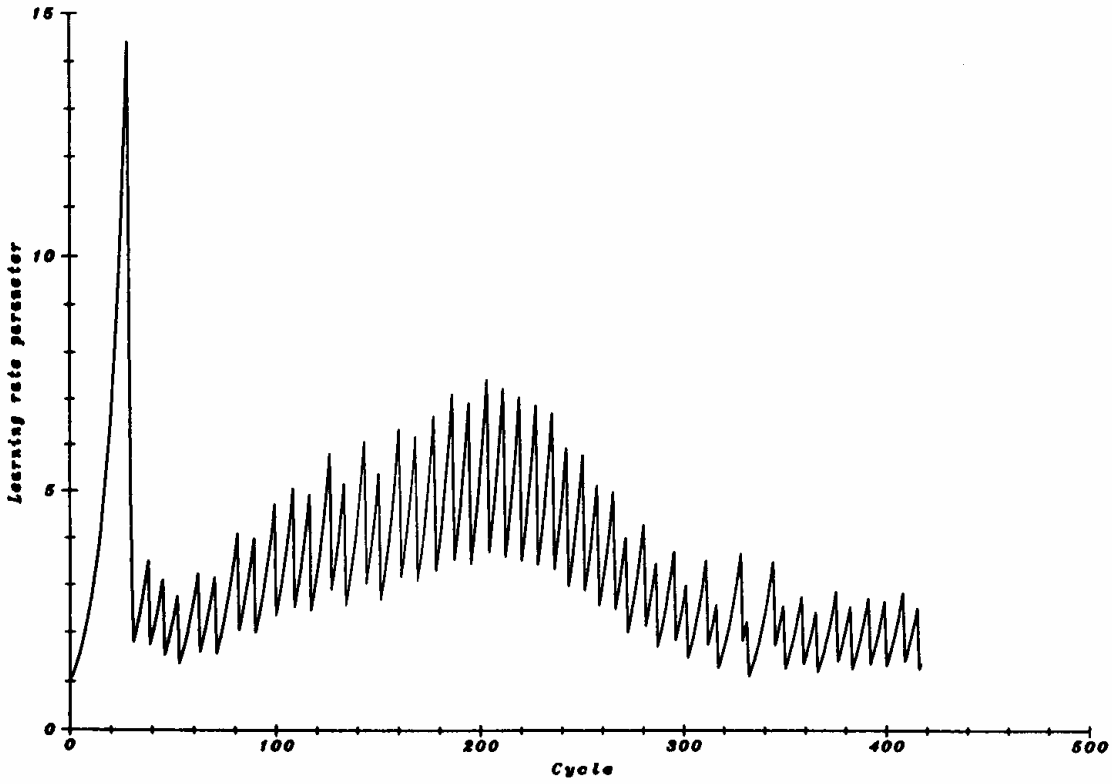

Figure 1: Example of learning rate behavior as a function of the iteration number for the "bold driver" network.

After experimenting on different problems with this apparently "quick and dirty" method, I have many indications that its performance (considering both the number of iterations required and the quality of the local minimum found) is close and usually better than that obtainable by optimizing a learning rate that is to remain fixed during the procedure. Besides the momentum term, there are now no learning parameters to be tuned by the user on each problem. The given values for ρ and σ can be fixed once and for all, and moreover, performance does not depend critically on their choice, provided that the heuristic guidelines given above are respected.

3. Second method: conjugate gradient with inexact linear searches

Let us define the following vectors: $\mathbf{g}_n = \nabla E(\mathbf{w}_n)$, $\mathbf{p}_n = \mathbf{w}_n - \mathbf{w}_{n-1}$, and $\mathbf{y}_n = \mathbf{g}_n - \mathbf{g}_{n-1}$.

Conjugate gradient methods [5] are iterative methods to generate successive approximations to the minimizer $\mathbf{w}^*$ of a function $E(\mathbf{w})$ in the following way:

$$\mathbf{d}_0 = -\mathbf{g}_0 \tag{3.1}$$

$$\mathbf{d}_n = -\nabla E(\mathbf{w}_n) + \beta_n \mathbf{d}_{n-1} \tag{3.2}$$

where

$$\beta_n = \left(\frac{\mathbf{y}_n \cdot \mathbf{g}_n}{\mathbf{y}_n \cdot \mathbf{d}_n}\right) \tag{3.3}$$

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \epsilon_n \, \mathbf{d}_n \tag{3.4}$$

where

$$\epsilon_n = \min_{\epsilon} E(\mathbf{w}_{n-1} + \epsilon \, \mathbf{d}_n) \tag{3.5}$$

If the function $E(\mathbf{w})$ is quadratic in an N-dimensional space ($E(\mathbf{w}) = \mathbf{c}^T\mathbf{w} + 1/2 \, \mathbf{w}^T G\mathbf{w}$, where G is a positive definite symmetric matrix), this method is guaranteed to converge to the minimum in at most $N+1$ function and gradient evaluation.

Two critical issues have to be considered in applying conjugate gradient methods to backpropagation learning. First, computation required during the exact one-dimensional optimization implied by equation (3.5) is expensive because every function evaluation involves a complete cycle of pattern presentation and error backpropagation; therefore, efficient approximate one-dimensional optimization have to be used. Second, since the function in this case in not quadratic, the convergence properties of the method are not assured *a priori* but depend on the degree that a local quadratic approximation can be applied to the energy surface.

Shanno [13] reviews several conjugate gradient methods and suggests one method using inexact linear searches and a modified definition of the search direction that "substantially outperforms known conjugate gradient methods on a wide class of problems." In the suggested strategy, the search direction for the nth iteration is defined as

$$\mathbf{d}_n = -\mathbf{g}_n + A_n \mathbf{p}_n + B_n \mathbf{y}_n \tag{3.6}$$

Every N steps (N being the number of weights in the network) the search is restarted in the direction of the negative gradient.

The coefficients A_n and B_n are combinations of scalar products of the vectors defined at the beginning of this section, as follows:

$$A_n = -\left(1 + \left(\frac{\mathbf{y}_n \cdot \mathbf{y}_n}{\mathbf{p}_n \cdot \mathbf{y}_n}\right)\right)\left(\frac{\mathbf{p}_n \cdot \mathbf{g}_n}{\mathbf{p}_n \cdot \mathbf{y}_n}\right) + \left(\frac{\mathbf{y}_n \cdot \mathbf{g}_n}{\mathbf{p}_n \cdot \mathbf{y}_n}\right) \tag{3.7}$$

$$B_n = \left(\frac{\mathbf{p}_n \cdot \mathbf{g}_n}{\mathbf{p}_n \cdot \mathbf{y}_n} \right) \tag{3.8}$$

Correction of the search direction based on previous steps is in part reminiscent of the use of a *momentum* term introduced in [11], with the added feature that a definite prescription is given for the choice of the various factors.

The one-dimensional minimization used in this work is based on quadratic interpolation and tuned to backpropagation, where in a single step both the energy value and the negative gradient can be efficiently obtained. Details on this step are contained in Appendix B.

Results of numerical experiments indicate that the above method, while requiring only minor changes to standard backpropagation, is capable of reducing the number of steps to convergence by approximately one order of magnitude on a large set of problems (tests have been done for small networks with up to 50 weights). Two example problems and the obtained results are described in the two following sections.

4. Test: the dichotomy problem

This problem consists in classifying a set of randomly generated patterns in two classes. It has been demonstrated [1] that an arbitrary dichotomy for any set of N points in general position in d dimensions *can* be implemented with a network with one hidden layer containing $[N/d]$ neurons. This is in fact the smallest such net as dichotomies which cannot be implemented by any net with fewer units can be constructed. In this test, the pattern coordinates are random values belonging to the [0-1] interval.

A dichotomy problem is defined by the number of patterns generated. The dimension of the space and the number of inputs is two, the number of middle-layer units is $[N/2]$ by the above criterion and one output unit is responsible for the classification.

Simulation runs have been made starting from small random weights (to break symmetry), with maximum size r equal to 0.1. Correct performance is defined as coming within a margin of 0.1 of the correct answer. Results of the "bold driver" and the conjugate gradient methods are compared in figure 2.

The capability of the network does not avoid the problem of local minima. These points are detected in an approximate way by terminating the search when the modulo of the gradient or of the weight change becomes less than 10^{-6}. In fact, the results show that their number is increasing as a function of the dimension of the search space (i.e., the number of weights in the network). Results on 128 tests for each problem (changing the random number seed) are given in tables 1 and 2.

To obtain the *effective* speedup of the inexact conjugate gradient method, the average time required by a conjugate gradient step has been compared with that required by one backpropagation step. Results for these ratios

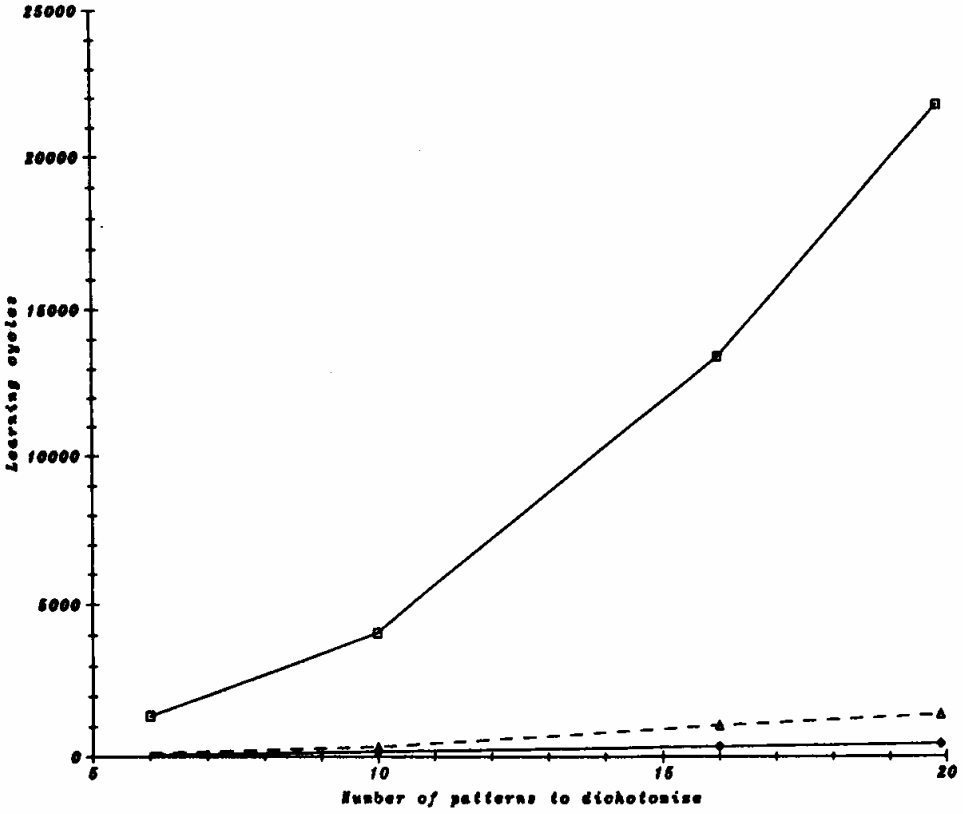

Figure 2: Performance comparison: backpropagation with adaptive learning rate (squares) versus inexact conjugate gradient method (diamonds = number of cycles, triangles = effective cycles considering one backpropagation cycle as unit of measure).

Patterns	6	10	16	20
	Cases: cycles	Cases: cycles	Cases: cycles	Cases: cycles
correct	127: 1352 (2079)	113: 4067 (4730)	97: 13359 (12788)	91: 21641 (16711)
loc. min.	1: 41401 (0)	15: 15443 (17632)	31: 16304(16673)	37: 30145 (23425)

Table 1: Backpropagation with adaptive learning rate ("bold driver" method): average number of cycles and standard deviation for convergence (128 tests for each problem).

Patterns	6	10	16	20
	Cases: cycles	Cases: cycles	Cases: cycles	Cases: cycles
correct	106: 53 (88)	101: 175 (525)	95: 343 (504)	93: 418 (476)
loc. min.	22: 292 (453)	27: 1734 (5388)	33: 789 (1523)	35: 1125 (1111)

Table 2: Backpropagation with inexact conjugate gradient: average number of cycles and standard deviation for convergence to correct or suboptimal local minimum (128 tests for each problem).

Patterns	6	10	16	20
ratio	1: 2.27	1: 1.89	1: 3.02	1: 3.32
speedup	11.23	12.29	12.89	15.59

Table 3: Comparison of the two "accelerated" backpropagation methods: timing ratios for a single cycle and resulting speedup.

together with the effective speedups are in table 3. A more detailed study on the scaling of the speedup with respect to problem size is currently under investigation.

5. Test: the parity function

Recently, Tesauro and Janssens [15] measured optimal averaging training times and optimal parameters settings for standard backpropagation with momentum term. In order to benchmark the two new proposed methods, the same network is used (n input units, $2n$ hidden units, one output) and weights are initialized randomly using the same scale parameter r_{opt} and momentum rate parameter α_{opt} as those given in [15].

The results of 100 simulations for each problem up to n equal to four show first, that backpropagation with adaptive learning rate produces results that are close (and even notably better for $n = 4$) to those obtained by optimizing parameters in backpropagation with fixed learning rate, and second, that the inexact conjugate gradient method brings a sizable speedup on both previous methods. Visual and numerical displays of results are in figure 3, table 4, and table 5. Since the number of local minima is negligible in this case (approximately 1% of the cases), only data regarding correct convergence are shown.

6. Summary and discussion

Both proposed methods can be implemented with parallel hardware (for example, one can assign different sets of neurons to different processors, or, for large grain size MIMD, one can assign different patterns to be learned to different processors) and require only one global exchange of information for each backpropagation cycle, in order to choose the next learning rate and search direction.

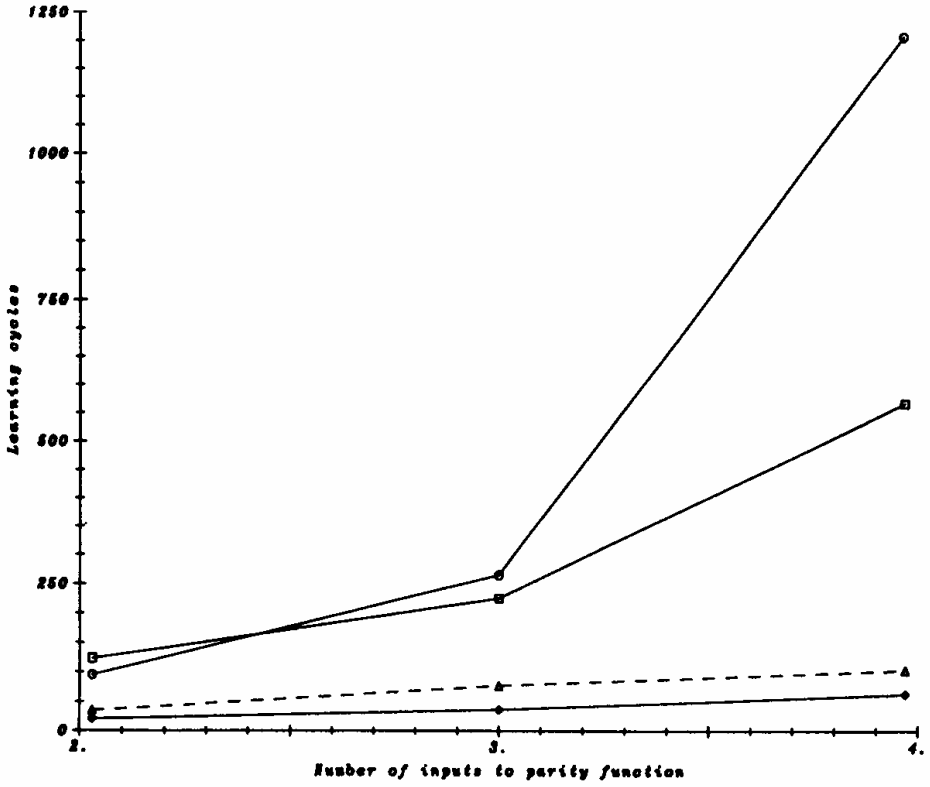

Figure 3: Performance comparison: standard backpropagation with optimal parameters from Tesauro and Janssens (circles), backpropagation with adaptive learning rate (squares), and inexact conjugate gradient method (diamonds = number of cycles, triangles = effective cycles considering one backpropagation cycle as unit of measure).

Inputs	2	3	4
	average cycles	average cycles	average cycles
backprop.	95 (?)	265 (?)	1200 (?)
bold driver	123 (190)	225 (98)	564 (299)
conj. grad.	20 (14)	36 (49)	62 (51)

Table 4: Timing comparison between standard backpropagation with optimal parameters (from Tesauro and Janssens) and the two methods suggested in the article.

Inputs	2	3	4
ratio	1: 1.72	1: 2.13	1: 1.65
speedup	2.76	3.45	11.73

Table 5: Comparison of optimized standard backpropagation and inexact conjugate gradient. Timing ratios for a single cycle and resulting speedup.

In conclusion, by relaxing the locality requirement for backpropagation, while maintaining most of its *parallelism*, one method ("bold driver") produces results close to the optimal ones (for fixed parameters), avoiding the user-driven optimization of parameters, while the second one (conjugate gradient with inexact linear searches) converges in a time that is typically an order of magnitude smaller than that required by standard backpropagation.

At present we are working on a parallel implementation of these methods in order to investigate performance for networks with a large number of weights.

Acknowledgments

This work was done as a research assistant of Geoffrey Fox and benefited in many ways from his advice. I thank Roy Williams for directing my attention to the conjugate gradient method, used in [18]. I am also pleased to acknowledge useful discussions with Edoardo Amaldi. Work supported in part by DOE grant DE-FG-03-85ER25009, the National Science Foundation with Grant IST-8700064 and by IBM.

Appendix A. One-dimensional minimization

Let us write $E(\epsilon)$ for $E(\mathbf{x}_{n-1} + \epsilon \mathbf{d}_n)$ where $\mathbf{d}_n$ has been defined in equation (3.6). First, $E(0)$ and $E(\epsilon = 4\epsilon_{n-1})$ are calculated.

If $E(\epsilon = 4\epsilon_{n-1})$ is greater or equal to $E(0)$, the parameter ϵ is divided by four until $E(\epsilon)$ is less than $E(0)$. Since $\mathbf{d}$ is guaranteed to be a descent direction this point will be found. Then, the minimizer $\epsilon_{\min}$ of the parabola going through the three points is found. The process is then repeated with the three points obtained after substituting $\epsilon_{\min}$ for one of the three previous points, in order to reobtain the configuration with the function value at middle point less than that at either end. The process continues until the difference in the last two approximation to the minimum value is less than 10^{-6}.

On the contrary, if $E(\epsilon = 4\epsilon_{n-1})$ is less than $E(0)$, the parameter ϵ is multiplied by four until $E(\epsilon)$ is greater than $E(0)+\epsilon E'(0)$ (to assure existence of a minimum in the quadratic minimization step). If this is found, the final ϵ is set either to the quadratic minimizer of the parabola through $E(0)$ and $E(\epsilon)$ with initial derivative $E'(0)$ or to $4\epsilon_{n-1}$, depending on the minimum value of the energy function for these two points. If this is not found after a reasonable number of trials (5 in our case), the final ϵ is set to $4\epsilon_{n-1}$.

The efficiency of the method is due to the fact that only a very limited number of iterations are actually done in the two cases. Furthermore, in the second case, the derivative $E'(0)$ is obtained rapidly with the scalar product of $\mathbf{d}_n$ and $\mathbf{g}_n$, which in turn are found together with the value $E(0)$ during the last backpropagation step.

References

[1] E.B. Baum, "On the capabilities of multilayer perceptrons," *Journal of Complexity*, **4** (1988) 193–215.

[2] A. Borsellino and A. Gamba, "An outline of a mathematical theory of PAPA," *Nuovo Cimento Suppl. 2*, **20** (1961) 221–231.

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

[4] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors* (Prentice Hall, New Jersey, 1988).

[5] P.E. Gill, W. Murray, and M.H. Wright, *Practical Optimization* (Academic Press, 1981).

[6] R.P. Gorman and T.J. Seinowski, "Analysis of hidden units in a layered network trained to classify sonar targets," *Neural Networks*, **1** (1988) 75–89.

[7] A. Lapedes and R. Farber, "A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition," *Physica*, **22D** (1986) 247–259.

[8] A. Lapedes and R. Farber, "Nonlinear signal processing using neural networks: prediction and system modeling," Los-Alamos Preprint LA-UR-87-1662.

[9] T.J. Seinowski and C.R. Rosenberg, "Parallel networks that learn to pronounce english text," *Complex Systems*, **1** (1987) 145–168.

[10] R.A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, **1** (1988) 295–307.

[11] D.E. Rumelhart and J.L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations* (MIT Press, 1986).

[12] D.E. Rumelhart and J.L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 2: Psychological and Biological Models* (MIT Press, 1986).

[13] D.F. Shanno, "Conjugate gradient methods with inexact searches," *Mathematics of Operations Research*, **3(3)** (1978) 244–256.

[14] G. Tesauro, "Scaling relationship in backpropagation learning: dependence on the training set size," *Complex Systems*, **1** (1987) 367–372.

[15] G. Tesauro and B. Janssens, "Scaling relationship in backpropagation learning," *Complex Systems*, **2** (1988) 39–44.

[16] T.P. Vogl, J.K. Mangis, A.K. Rigler, W.T. Zink, and D.L. Alkon, "Accelerating the convergence of the backpropagation method," *Biological Cybernetics*, **59** (1988) 257–263.

[17] P.J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, 1 (1988) 339–356.

[18] R.D. Williams, "Finite elements for 2D elliptic equations with moving nodes," Caltech C3P Report, (1987) 423 (available from Concurrent Computation Project, Caltech, Pasadena, CA 91125).