On Training Neural Nets through Stochastic Minimization

Roberto Brunelli
Istituto per la Ricerca Scientifica e Tecnologica
I-38050 Povo, Trento, ITALY

Abstract—**The revival of multilayer neural networks in the mid 80's originated from the discovery of the back-propagation technique as a feasible training procedure. In spite of its shortcomings, it is probably the most widespread technique for training feedforward nets. In recent years, several deterministic methods more efficient than back-propagation have been proposed. In this paper a stochastic minimization algorithm, Iterated Adaptive Memory Stochastic Search, is described which does not use gradient information and is found to perform better than back-propagation on the encoder and parity problems[1].**

Keywords: stochastic optimization, learning algorithms, back-propagation

1. Introduction

Learning from examples, the problem which neural networks were created to solve, is one of the most important research topics in the AI community. A possible way to formalize learning from examples is to hypothesize the existence of a function that captures the underlying mapping, thereby enabling generalization. The problem of learning is then replaced by that of function reconstruction from sparse data. Within this general framework the main points of interest are the representational power of a given network architecture and the procedures for obtaining the optimal network parameters. The first point, i.e., which functions can be actually approximated by a given network architecture and how good the approximation can be, has received much attention [12], [8], [15], [16]. The main results state that, under commonly satisfied assumptions, multilayer feedforward networks are a class of universal approximators. In this paper the second problem will be addressed, i.e., which strategies can be used to compute network parameters which are optimal according to some given criterion. A new, adaptive, stochastic minimization algorithm is presented which does not use gradient information. The performance of the algorithm on the encoder and parity problems is compared to that of back-propagation and to other training algorithms reported in the literature.

2. A brief review of back-propagation

The revival of multilayer neural networks in the mid 80's originated from the discovery of the back-propagation technique as a feasible training procedure. Let us briefly review the back-propagation algorithm for a feedforward, fully connected neural network [24] with $n+1$ layers and N_i units per layer. The output of unit i of layer j is given by:

$$o_{i,j} = \sigma(\sum_k w_{i,j,k} o_{k,j-1}) \tag{1}$$

where σ is usually the logistic function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

Training is realized by minimizing the overall square error defined by:

$$E(\mathbf{w}) = \frac{1}{2} \sum_p \|\mathbf{T}_p - \mathbf{O}_p(\mathbf{w})\|^2 \tag{3}$$

where p runs over the set of different input patterns and $\mathbf{T}_p, \mathbf{O}_p$ represent the target value and its net approximation respectively. Each pass through the entire training set, to compute $E(\mathbf{w})$ and related quantities, will be referred to as an *epoch*. Note that minimizing the total square error corresponds to updating the network weights by epoch (batch learning) and not by case (on-line learning). The update formula for the weights is:

$$\delta w_{i,j,k} = -\alpha \frac{\partial E}{\partial w_{i,j,k}} \tag{4}$$

It turns out that the gradient can be efficiently computed by backward propagating the error from the output level through the network layers (this is in fact the origin of the name). One of the most popular ways to improve the convergence is the introduction of a momentum term [24] into eqn (4):

$$\delta w^{(l)}_{i,j,k} = -\alpha \frac{\partial E}{\partial w_{i,j,k}} + \mu\, \delta w^{(l-1)}_{i,j,k} \tag{5}$$

where α represents the *learning rate* and the superscript represent the epoch. As noted in [30], eqn (5) can be reformulated as a first-order difference equation:

$$\mathbf{w}^{(l+1)} = \mathbf{w}^{(l)} - \alpha \sum_{j=0}^{l} \mu^{l-j} \nabla E(\mathbf{w})^{(j)} \tag{6}$$

The use of a momentum term corresponds then to taking into account an exponentially damped memory of previous steps. The choice of the parameters α and μ is essentially heuristic and they keep the same value during the training of the network. The cost of the trial runs which are usually necessary for selecting appropriate values for α, μ should be included in the cost of using back-propagation to train a network for a specific task. Besides the addition of momentum, other heuristics can be used to speed up back-propagation among which we can find the re-use of already computed gradient information [14] and the adaptive choice of the learning parameter α [29], [17], [31]. The

The author would like to thank Prof. T. Poggio, B. Caprile and C. Furlanello for helpful discussions.

[1]This research was done within MAIA, the integrated Artificial Intelligence project under development at IRST [22]

performance of back-propagation can be greatly influenced by the choice of the learning and momentum parameters. Another point of concern is given by the possibility of getting trapped into a local minimum. Given a learning task, it is usually unknown whether bad local minima exist and the problem is usually solved by iterated application of the learning procedure from different starting configurations. In spite of these shortcomings, back-propagation is probably the most widespread technique for training feedforward nets.

3. Stochastic Minimization

As briefly noted before, the problem of learning can be reduced to that of function reconstruction from sparse data. Usually a parametric set of functions $\mathbf{G}(\mathbf{x}, \mathbf{\Pi})$ is selected and an optimal parameter set $\mathbf{\Pi}_0$ is chosen using a *Least Mean Square* criterion:

$$\begin{aligned} \mathbf{\Pi}_0 \quad &: \quad E(\mathbf{\Pi}_0) = \min_{\mathbf{\Pi}} E(\mathbf{\Pi}) \qquad (7) \\ E(\mathbf{\Pi}) \quad &= \quad \frac{1}{2} \sum_i (\mathbf{T}(\mathbf{x}_i) - \mathbf{G}(\mathbf{x}_i, \mathbf{\Pi}))^2 \end{aligned}$$

where $\{(\mathbf{x}_i, \mathbf{T}(\mathbf{x}_i))\}$ represent the available examples. Back-propagation is a technique to minimize $E(\mathbf{\Pi})$ using gradient descent. Several other algorithms for training multilayer feedforward networks have been proposed and compared to back-propagation in the last few years [30], [29], [17], [19], [3]. Simulated annealing [20], [18] and genetic algorithms [11] represent two alternative optimization techniques based on random search. An important application of simulated annealing is the learning algorithm of the Boltzmann machine [1]. Genetic algorithms have also been used in the field of machine learning and of function optimization [10], [32].

Random search techniques, first proposed by [2], can be competitive, with respect to methods based on gradient and/or Hessian information, whenever this additional information can not be computed or the presence of local minima favours extensive search versus precision. Many algorithms have been proposed so far and the convergence properties have been investigated both theoretically [25], [27], [28] and experimentally [26], [27], [3], [6], [7], [28], [5]. In this paper a stochastic algorithm is introduced, called IAMSS (Iterated Adaptive Memory Stochastic Search) and based on the use of an adaptive memory. As no gradient information needs to be computed, the algorithm can be used to train networks whose activation functions are not differentiable: they do not even need to be continuous. A related observation is that a network can now be trained using a non continuous error function, such as the classification error, thereby overcoming some of the limitations of *Least Mean Square Learning*.

The next section will describe the basic algorithm. The description is based mainly on the work of [5] which presents the algorithm as a local search method applied to several minimization tasks. A heuristic restart criterion is then introduced, so that the algorithm becomes, under suitable assumptions, a global search method [27]. The performance of the algorithm is finally compared to that of back-propagation, conjugate gradient and steepest descent on the encoder and parity problems.

3.1 Stochastic search with adaptive memory

The algorithm is based on a modified version [28], [5] of a nondeterministic minimization algorithm proposed by [6] and [7]. The local random search presented by [5] is characterized by the introduction of an adaptive memory. Stochastic minimization algorithms usually make little use of information on the local structure of the function to be minimized. This approach, while justified when the function has a complicated microstructure, reduces the search efficiency when the *landscape* is smooth but anisotropic. The introduction of an adaptive memory enables the algorithm to decide by itself how much of the information gathered during the minimization process can be successfully used to guide the search. The algorithm presented in this paper is further characterized by the introduction of a heuristic stop criterion, based on the minimization trend. This permits to restart the search for the global minimum location from a new point, randomly distributed over the domain of the function to be minimized, when no further improvements are to be expected from the current search.

The description will be limited to the coordinatewise search strategy which exhibits better convergence near a local minimum [28].

Given a *probe* point $\mathbf{x}$, representing in vector form the set of minimization parameters, each component x_i is modified in turn by the addition of a random value uniformly distributed in the region $[-r_i, -r_i/2] \bigcup [r_i/2, r_i]$. After a single component is modified, if the function decreases by moving to the perturbed position, the new point is taken as the estimate of the minimum location and r_i is doubled. If the function does not decrease, a new point is considered by adding to the original point the reversed perturbation (*double shot*). Should this succeed, the probe is moved and r_i is doubled. In case of failure, the point is left unmoved and r_i is halved. In the following, a perturbation cycle over all of the coordinates will be referred to as a *sweep*.

The memory mechanism of the algorithm is rather simple: jumps of the probe point are stored in a circular buffer, the memory, and a linear combination of them is used as an hint for an additional jump. In particular, let us introduce normalized time t defined as

$$t = \frac{n_{fe}}{P} \qquad (8)$$

where n_{fe} is the number of function evaluations and P the number of minimization parameters. The number of function evaluations is the overall number of times the objective function, i.e., the total square error, is computed and corresponds to the number of epochs. This includes unsuccessful trial perturbations. The number of minimization parameters corresponds to the overall number of weights and thresholds of the semilinear nodes in the network to be trained. This number represents the dimensionality of the domain of the function to be minimized.

The successful perturbations (hereafter displacements) of the probe point are stored into the circular memory buffer according to the following schedule. After testing all of the coordinates, the displacements δ_k stored in the buffer are added using Gaussian weights:

$$\Delta(t) = \tilde{\alpha}\, \frac{1}{W} \sum_{k=0}^{n-1} e^{-(t-t_k)^2/\tilde{\mu}^2} \delta_k \qquad (9)$$

where:

$$W = \sum_{k=0}^{n-1} e^{-(t-t_k)^2/\tilde{\mu}^2}$$

and $n-1$ represent the number of perturbation cycles over the coordinates (modulus the size of the memory buffer). The memory of the algorithm is then characterized by two parameters: its *depth* $\tilde{\mu}$ and its *amplification* $\tilde{\alpha}$. If $\Delta(t)$ results in a successful perturbation the amount of memory is *increased*:

$$\tilde{\alpha} \leftarrow 2\tilde{\alpha} \quad (10)$$

$$\tilde{\mu} \leftarrow \frac{3}{2}\tilde{\mu} \quad (11)$$

otherwise it is decreased:

$$\tilde{\alpha} \leftarrow \max(0.1, \frac{\tilde{\alpha}}{4}) \quad (12)$$

$$\tilde{\mu} \leftarrow \max(2.0, \frac{\tilde{\mu}}{4}) \quad (13)$$

The displacement of the probe point, computed by adding the displacements along each of the coordinates and the perturbation $\Delta(t)$ if successful, is marked with current time and stored into the memory buffer as δ_n. The sign of the i-th component of $\Delta(t)$ is used to bias the perturbation of x_i in the subsequent sweep over the coordinates.

The control parameters $\tilde{\mu}$ and $\tilde{\alpha}$ are independent of the problem dimensionality (due to the use of a *normalized* time) and can be considered constants of the algorithm. In the reported experiments the size of the memory buffer was limited to the number of network parameters:

$$P = \sum_{i=1}^{n} (N_{i-1}+1)N_i \quad (14)$$

with N_0 the number of inputs to the network. We can draw a parallel with back-propagation with momentum: the (adaptive) scope of the random search represents the learning rate while the vector $(\tilde{\alpha}, \tilde{\mu})$ which describes the memory represents the momentum term. The described method presents some similarity to the enhanced variants of back-propagation proposed by [29] and [17].

3.2 Stop-and-Restart Heuristic

The optimization problem stated in eqn (7) should be considered, according to [9], as one of

Limited global optimization. *Given a functional f*

$$f : X \subset \mathbf{R}^n \longrightarrow \mathbf{R}$$

find an element (a point) $\mathbf{x}_0 \in X$ *such that*

$$f(\mathbf{x}_0) \leq f(x) \ \textit{for all}\ \mathbf{x} \in X \quad (15)$$

This should be compared with the following, distinct, optimization problem:

Local optimization. *Given a functional f*

$$f : X \subset \mathbf{R}^n \longrightarrow \mathbf{R}$$

find an element (a point) $\mathbf{x}_0 \in X$ *such that*

$$f(\mathbf{x}_0) \leq f(x) \ \textit{for all}\ \mathbf{x} \in N_\delta(\mathbf{x}_0) \quad (16)$$

where

$$N_\delta(\mathbf{x}_0) = \{\mathbf{x} \in X | \ \|\mathbf{x}-\mathbf{x}_0\| < \delta, \delta > 0\}$$

in some (maybe, truncated) δ-neighborhood in X. Here $\|\cdot\|$ is a norm in the appropriate space.

If X and f are convex, local and limited global optimization coincide and can be solved with the same techniques. In general, X and f are not convex, and techniques such as gradient descent and conjugate gradient which can solve problems of local optimization cannot solve, directly at least, those of limited global optimization. However, if the set X is compact, any local algorithm $\mathcal{A}$ which stops after a finite time, can be used to solve a limited global optimization problem: it can be applied repeatedly starting from points uniformly distributed over X [27]. The algorithm described in the previous section can then be used to solve the optimization problem of eqn (7) after the introduction of a stopping criterion.

In the following paragraphs a *heuristic* stopping criterion, based on extensive experimentation with IAMSS, will be introduced. Let us focus on a local optimization problem. As reported by [5], the empirical convergence rate of the algorithm in the minimization of quadratic forms (even with strong conditioning numbers) turns out to be exponential:

$$y(t) - y_{min} = ae^{-bt} \quad (17)$$

where t is the number of functional evaluations divided by the dimensionality of the parameter space (normalized time), y_{min} and $y(t)$ represent the minimum value and the estimate of the minimum value of the functional to be minimized at time t respectively (e.g. the local minimum of the total square error and its estimate). If the functional to be minimized is analytic, its shape in the neighborhood of a local minimum can often be modelled by the quadratic term of its Taylor expansion. This implies that the estimate of the local minimum value should approach the true value with the above given trend. The stopping rule is based on the goodness of the exponential fit obtained by assuming that the minimum value so far represents the real minimum within precision ϵ_p.

Let $y(t_i)$ represent the functional value after testing the usefulness of memory at the end of the i-th sweep. The n most recent values are considered for fitting the exponential trend:

$$y(t) - \widehat{y}_{min}(t_i) = ae^{-bt} \quad (18)$$

where $\widehat{y}_{min}(t_i)$ is the estimated value of the local minimum:

$$\widehat{y}_{min}(t_i) = (1-\epsilon_p)\ y(t_i) \quad (19)$$

Optimal values for a, b are found using the least square method. The goodness of fit is estimated by the median of the absolute percentage error:

$$D = \text{median} \left\{ \left| \frac{(y(t_{i-k}) - \widehat{y}(t_{i-k}))}{(\widehat{y}(t_{i-k}) - \widehat{y}_{min}(t_i))} \right| \right\}_{k=0,\ldots,n-1} \quad (20)$$

where $\widehat{y}(t_{i-k})$ represent the estimated values. The algorithm is stopped and restarted when the fit error is less than a given threshold (which was taken to be 0.3 in all of the reported experiments). The reason for using the median is that it is a more robust estimator than the arithmetic average [13]. In the reported experiments n was set to 5

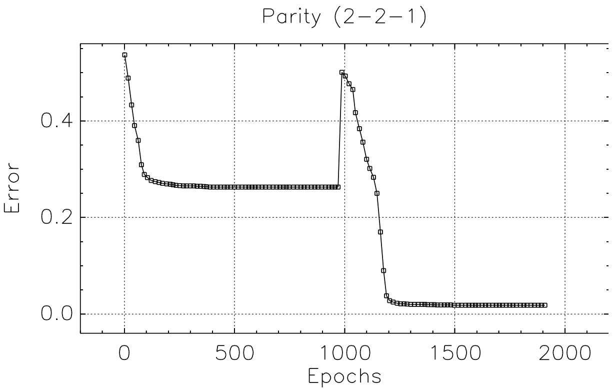

Fig. 1. A typical training for the parity 2-2-1 problem (XOR) using the IAMSS algorithm. The total square error is given as a function of the number of epochs.

thus making the test computationally inexpensive. The algorithm is also stopped and restarted when the maximum search scope $\max_i\{r_i\}$ becomes smaller than the square root of the precision at which computation is carried out [23]. Finally, the introduction of an upper bound on the number of functional evaluations guarantees that the local search algorithm eventually halts, thereby meeting the needs for a limited global optimization algorithm. A typical training is reported in Figure 1 for the XOR problem. The average square error on the training set is reported as a function of the number of epochs.

4. Comparison with gradient based techniques

As mentioned in the introduction, several comparison of back-propagation with more sophisticated techniques based on the use of gradient information have appeared in the literature (e.g. [30]). This section will refer mainly to the paper of [19] where the networks are trained by minimizing

$$E'(\mathbf{w}) = E(\mathbf{w}) + \frac{\gamma}{2}\|\mathbf{w}\|^2 \tag{21}$$

with $\mathbf{w}$ representing the network weights. The modified error measure proves to be helpful in learning boolean functions. In fact, for such functions the minima of $E(\mathbf{w})$ are located at infinity while those of $E'(\mathbf{w})$ are located within a compact region. The weights of the network are generated at random in an assigned interval $[-r, r]$. The algorithms are terminated when a constraint on the Maximum Bit Error[2] (MBE) is satisfied:

$$\text{MBE} < \tau \tag{22}$$

or the magnitude of the gradient becomes less than a given threshold ϵ. The termination condition was modified for the IAMSS algorithm to preserve its characteristics: the stopping condition on the magnitude of the gradient was replaced by a limit on the maximum number of iterations N_{max} (in the reported experiments $N_{max} = 3 \times 10^5$) and ϵ was used as a minimum noise level to supplement the exponential test for restarting the algorithm. Table I gives the values used in the training experiments.

The stochastic algorithm has been applied to the encoder and parity problems. The results obtained on these tasks together with those obtained by [19] are reported in Tables II and III for the following algorithms:

PR : Polak-Ribiere conjugate gradient;
SD : steepest descent;
BP : back-propagation;
IAMSS : iterated adaptive memory stochastic search.

As the average complexity of a single epoch varies among the considered algorithms, an approximate complexity has been computed for each of them and used to compute a normalized average number of epochs by setting the complexity of back-propagation to 1 (see Appendix for details). For all of the algorithms the *normalized number of expected epochs to solution* (NEES) is given, defined as the ratio of total epochs to successful trials multiplied by the complexity factor which is also reported in the tables. The results for the encoder problem are reported in Table II.

The performance of IAMSS on this *smooth* problem is comparable to that of back-propagation while Polak-Ribiere conjugate gradient exhibits the fastest convergence.

The results for the parity problem (see Table III) show that IAMSS reports the best success rate among the compared algorithms. While conjugate-gradient is found to be the most efficient, the stochastic algorithm compares favourably with steepest-descent and outperforms back-propagation. As already mentioned, the stochastic algorithm could be particularly useful when the error function (and/or the activation function of the network cells) is noncontinuous such as in a classification task, where the classification error is a more meaningful quantity than the approximation error of the characteristic function of a given

Parameter	Value
r	1
γ	10^{-4}
τ	0.1
ϵ_p	10^{-7}
ϵ	10^{-4}

TABLE I
Values of the control parameters of the experiment.

Encoder	Algorithm	Trials	Success (%)	NEES	Comp.
4-2-4	PR	100	100	25	0.68
	SD	100	100	38	0.66
	BP	100	100	180	1.00
	IAMSS	100	100	114	0.54
8-3-8	PR	100	100	43	0.63
	SD	100	100	123	0.63
	BP	100	100	555	1.00
	IAMSS	100	100	450	0.49
16-4-16	PR	100	100	75	0.62
	SD	100	100	355	0.62
	BP	100	100	990	1.00
	IAMSS	100	100	1527	0.47

TABLE II
Comparison of the algorithms on the encoder problem. The first column describes the structure of the network by giving the number of units in the input, hidden and output layers. NEES represents the normalized number of expected epochs to solution while the last column reports the normalized complexity facto.

[2] Defined as $MBE = \max_{pi} \frac{1}{2}|y_{pi} - \widehat{y}_{pi}(\mathbf{w})|^2$, with p running over the input patterns and i over the output coordinates.

Parity	Algorithm	Trials	Success (%)	NEES	Comp.
2-2-1	PR	100	72	124	0.76
	SD	100	80	596	0.69
	BP	100	78	14197	1.00
	IAMSS	100	100	659	0.64
4-4-1	PR	100	61	436	0.68
	SD	100	99	1534	0.66
	BP	100	71	48430	1.00
	IAMSS	100	100	2246	0.54
8-8-1	PR	16	50	1735	0.65
	SD	6	-	$> 10^4$	0.65
	BP	2	-	$> 10^5$	1.00
	IAMSS	20	85	66195	0.51

TABLE III

Comparison of the algorithms on the parity problem. The first column describes the structure of the network by giving the number of units in the input, hidden and output layers. NEES represents the normalized number of expected epochs to solution while the last column reports the normalized complexity factor.

Problem	Starts	Restarts
Encoder (4-2-4)	100	0
Encoder (8-3-8)	100	0
Encoder (16-4-16)	100	0
Parity (2-2-1)	100	66
Parity (4-4-1)	100	11
Parity (8-8-1)	20	19

TABLE IV

Number of restarts made by the stochastic algorithm.

class of points. In this case a mixed error function E'' could be used:

$$E'' = E_{class.} + \lambda\, E' \tag{23}$$

where λ is a weighting factor. A similar error function was used in [4] where an HyperBF network [21] was used to discriminate gender from a geometrical description of facial features. Using the classification error only would make the problem much harder by introducing regions with no local information on how minimization could proceed. The number of restarts made by the stochastic algorithm in the different tasks is reported in Table IV and can be used as a qualitative estimate of the difficulty posed by the presence of local minima.

5. Conclusions

In this paper a novel, stochastic, minimization algorithm characterized by the use of an adaptive memory is introduced and applied to the training of fully connected feed-forward neural networks. The performance of the proposed algorithm is compared with that of back-propagation, conjugate gradient and steepest descent on the encoder and parity problems. The results obtained show that the algorithm is superior to back-propagation in the tasks considered and that it exhibits the best success rate, among the compared algorithms, on the parity problem.

As no gradient information needs to be computed, neural networks with non continuous activation and error functions can now be trained, overcoming some of the limitations exhibited by previous training procedures. This is expected to be particularly useful in classification tasks where a *natural* error measure is given by the number of misclassified patterns.

Appendix

The computational cost of a single epoch depends on the algorithm. It is then necessary to estimate this cost for each of the algorithms to make the comparison fair. The computation of the (approximated) normalizing factors relies on the following assumptions:

1. the average number of iterations used for minimizing along the direction of gradient is set to 2.5 [19];
2. the complexity of a single arithmetic operation is set to 1;
3. the complexity of evaluating $\sigma(x)$ or of generating a random number are set to 5;
4. the average number of epochs between two successive tests of memory usefulness in the stochastic algorithm is set to $1.5P$;
5. the computation is sequential.

The complexity of a forward sweep of the network (the computation of the network output for all of the available patterns plus the cumulative square error) is given by:

$$C_F = N_p \left[3N_n + \sum_{i=1}^{n} (2N_{i-1} + 5) N_i \right] \tag{24}$$

while the complexity of the gradient computation is given by:

$$\begin{aligned} C_G &= (N_p - 1)P + \\ &\quad + N_p \left\{ (3 + N_{n-1}) N_n + \sum_{i=1}^{n-1} \left[N_{i-1} + 2(N_{i+1} + 1) \right] N_i \right\} \end{aligned} \tag{25}$$

The approximate complexity of an average epoch for the compared algorithms is given by the following expressions:

PR :

$$C_{PR} = \frac{[C_F + C_G + 7P + 2.5(C_F + 2P) - 1]}{3.5} \tag{26}$$

SD :

$$C_{SD} = \frac{[C_F + C_G + 2.5(C_F + 2P)]}{3.5} \tag{27}$$

BP :

$$C_{BP} = (C_F + C_G + 2P) \tag{28}$$

IAMSS :

$$C_{IAMSS} = \frac{[P(1.5C_F + 2P + 13.5) + C_F + 5]}{1.5P} \tag{29}$$

and normalized complexity is obtained by setting $C_{BP} = 1$. The comparison of different algorithms should take into account both the number of *steps* and their *complexity*. This observation leads to the notion of normalized number of expected epochs to solution (NEES), used in the paper to measure the learning time for problems with bad local minima.

References

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] R. L. Anderson. Recent Advances in Finding the Best Operating Conditions. *J. Amer. Stat. Assoc*, 48:789–798, 1953.

[3] N. Baba. A new approach for finding the global minimum of error function of neural networks. *Neural Networks*, 2:367–373, 1989.

[4] R. Brunelli and T. Poggio. HyperBF Networks for Gender Classification. In *Proc. DARPA Image Understanding Workshop*, pages 311–314, 1992.

[5] R. Brunelli and G. Tecchiolli. Stochastic minimization with adaptive memory. Technical Report 9211-14, I.R.S.T, 1992. To appear on *Journal of Computational and Applied Mathematics*.

[6] B. Caprile and F. Girosi. A Nondeterministic Minimization Algorithm. A.I. Memo No. 1254, Massachusetts Institute of Technology, 1990.

[7] B Caprile, F. Girosi, and T. Poggio. A Nondeterministic Method for Multivariate Functions Minimization. In E. R. Caianiello, editor, *Parallel Architectures and Neural Networks*, pages 37–44. World Scientific, 1991.

[8] G. Cybenko. Approximation by superposition of a sigmoidal function. *Math. Control Signals Systems*, 2:303–314, 1989.

[9] E. A. Galperin. Problem-method classification in optimization and control. *International Journal of Computers and Mathematics with Applications*, 21:1–6, 1991.

[10] D. E. Goldberg. *Genetic Algorithms*. Addison-Wesley, Reading, MA, USA, 1989.

[11] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[12] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2:359–366, 1989.

[13] P. J. Huber. *Robust Statistics*. Wiley, 1981.

[14] D. R. Hush and J. M. Salas. Improving the Learning Rate of Back-Propagation with the Gradient Reuse Algorithm. In *IEEE International Conference on Neural Networks*, 1988.

[15] Y. Ito. Approximation of functions on a compact set by finite sums of a sigmoid function without scaling. *Neural Networks*, 4:817–826, 1991.

[16] Y. Ito. Approximation of continuous functions on R^d by linear combinations of shifted rotations of a sigmoid function with and without scaling. *Neural Networks*, 5(1):105–115, 1992.

[17] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[19] A. H. Kramer and A. Sangiovanni-Vincentelli. Efficient parallel learning algorithms for neural networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 40–48. Morgan Kaufmann, 1989.

[20] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[21] T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[22] T. Poggio and L. Stringa. A Project for an Intelligent System: Vision and Learning. *International Journal of Quantum Chemistry*, 42:727–739, 1992.

[23] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

[24] D. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, 1987.

[25] G. Schrack and M. Choit. Optimized Relative Step Size Random Searches. *Mathematical Programming*, 10:230–244, 1976.

[26] G. Shrack and N. Borowski. An Experimental Comparison of Three Random Searches. In F. Lootsma, editor, *Nonlinear Methods for Nonlinear Optimization*. Academic Press, 1972.

[27] F. J. Solis and R. J-B. Wets. Minimization by Random Search Techniques. *Mathematics of Operations Research*, 6:19–30, 1981.

[28] G. Tecchiolli and R. Brunelli. On random minimization of functions. *Biological Cybernetics*, 65:501–506, 1991.

[29] T. P. Vogl and al. Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, 59:257–263, 1988.

[30] R. L. Watrous. Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization. In *IEEE First International Conference on Neural Networks*, 1987.

[31] M. K. Weir. A Method for Self-Determination of Adaptive Learning Rates in Back Propagation. *Neural Networks*, 4:371–379, 1991.

[32] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, 14:347–361, 1990.

Symbols

o_{ij}	output of unit i of layer j
w_{ijk}	weight connecting the i-th unit of the j-th layer to the k-th unit of the previous layer
σ	sigmoidal activation function
$\mathbf{w}$	a vector representing the network weights
$\mathbf{T}_p(\mathbf{w}), \mathbf{O}_p(\mathbf{w})$	target value and network output corresponding to the p-th input pattern
N_i	number of network units in the i-th layer
P	number of parameters in the network
E, E'	total square error and total square error plus a term corresponding to the length of the weight vector
γ	weight given to the norm of $\mathbf{w}$ in the computation of E'
E''	a non continuous error measure given by the sum of E' and the number of misclassified patterns
λ	weight given to E' in the computation of E''
$\nabla E(\mathbf{w})$	gradient of $E(\mathbf{w})$
α, μ	learning rate and momentum term used by back-propagation
$\mathbf{G}(\mathbf{x}, \mathbf{\Pi})$	set of functions acting on $\mathbf{x}$ and specified by parameters $\mathbf{\Pi}$
$\{(\mathbf{x}_i, \mathbf{T}(\mathbf{x}_i))\}$	training examples
r_i	maximum perturbation allowed for the i-th component
n_{fe}	number of function evaluations
t	normalized time
δ_k	displacement of probe point after the k-th perturbation sweep over the coordinates
$\tilde{\alpha}, \tilde{\mu}$	amplification and depth of adaptive memory
$\Delta(t)$	temptative displacement of the probe point computed using the adaptive memory
$\underline{y}(t_i)$	the function value after testing the usefulness of memory at the end of the i-th sweep
$\widehat{y}_{min}(t_i)$	is the estimated value of the local minimum
D	goodness of the exponential fit
τ	maximum bit error
ϵ_p	square root of machine arithmetic precision
ϵ	minimum magnitude of perturbation and gradient
C_F, C_G	complexity of a forward sweep and a gradient computation
C_{PR}	complexity of Polak-Ribiere conjugate gradient
C_{SD}	complexity of steepest descent
C_{BP}	complexity of back-propagation
C_{IAMSS}	complexity of Iterated Adaptive Memory Stochastic Search