Adaptive Training of Multilayer Neural Networks Using A Least Squares Estimation Technique

Stefanos Kollias and Dimitris Anastassiou*

Department of Electrical Engineering

Columbia University, New York, NY 10027 USA

* On leave from the Department of Electrical Engineering

National Technical University of Athens, 15773 Greece

ABSTRACT

A new learning algorithm for the adaptive training of feedforward neural networks is presented, based on an efficient implementation of the Marquardt-Levenberg least squares optimization technique. The algorithm has nice convergence properties, which are superior to those of the backpropagation-momentum learning technique.

I. Introduction

The error criterion which is generally used in the training of neural networks is the minimization of a sum of squares function. For a network with M outputs and after the presentation of K pairs of input and desired output patterns, this function is

$$E_K = \frac{1}{2} \sum_{r=1}^{K} \sum_{m=1}^{M} [d(m) - y(m)]_r^2 \tag{1}$$

where the symbols d and y denote the desired and actual outputs of the network, and the subscript r shows their dependence on the rth input presentation.
When an input pattern is presented, it propagates forward through the network and each neuron computes its output value using the prespecified set of its input parameters. Let us consider the case of a network consisting of L layers, the lth of which contains $(N_l - 1)$ neurons. Then each neuron computes its output according to the following equations for $l = 1, L$ and $j = 1, N_l - 1$

$$x_j^l = f\left\{ \sum_{n=1}^{N_{l-1}-1} (w_{nj}^l \, x_n^{l-1}) \; - \; \theta_j^l \right\} \tag{2}$$

where x_j^l denotes the output of the jth neuron belonging to the lth layer and w_{nj}^l is the weight of the connection between the nth neuron of the (l-1) layer and the jth neuron of the lth layer. The Lth layer is the output layer, while layer zero is the input layer. Thus N_L and N_0 are equal to M and N respectively, while x_i^L and x_i^0 denote the output $y(i)$ and input $x(i)$ sequences. The function f is usually a sigmoid and θ_j^l is the threshold of the jth neuron of the lth layer, which controls the neuron's output when there are no signals to its input. The thresholds are generally treated as weights that connect an input unit, which is always *on*, i.e. its value is always one, to the neuron.

In the following Eq.(1) is used to formulate a non linear least squares minimization problem and various unconstrained optimization techniques are discussed for its solution,

including the well-known backpropagation [1]-[2] and Newton type [3]-[4] learning algorithms. A technique is then derived, which has major advantages over the backpropagation algorithm and is computationally efficient, compared to the usual quasi-Newton methods.

II. The Minimization Problem

Let us define the vector $\mathbf{w}_i^l$ of all input weights to the ith neuron of layer l

$$\mathbf{w}_i^l = [w_{1i}^l \cdots w_{(N_l-1)i}^l \; \theta_i^l]^T \tag{3}$$

and let the vector $\mathbf{x}^l$ contain the outputs of all neurons in layer l and one more that is always equal to one

$$\mathbf{x}^l = [x_1^l \cdots x_{N_l-1}^l \; 1]^T \tag{4}$$

Let us also define a vector $\mathbf{w}$ of all the parameters $\mathbf{w}_i^l$ of the network. A necessary condition at a global or local minimum of (1) is that the derivatives of this function with respect to $\mathbf{w}$ be zero. Forming these derivatives we get the following system of non-linear equations for $l = 1, L$ and $i = 1, N_l$

$$\mathbf{f}_i^l = \sum_{r=1}^{K} \sum_{m=1}^{M} [d(m) - y(m)]_r^T [-\frac{\partial y(m)}{\partial \mathbf{w}_i^l}]_r \;\; = \;\; 0 \tag{5}$$

where the dimension of vector $\mathbf{f}_i^l$ is N_l .
A well known iterative technique for the solution of such a system is the Newton's method [5]. One way to apply this method is to use a Taylor expansion of each of the above equations with respect to a value of $\mathbf{w}_0$, which is assumed to be in the neighbourhood of the minimum of Eq.(1). By keeping only the linear terms of this expansion and dropping the higher order ones, the problem reduces to that of finding the required additive correction terms $\Delta\mathbf{w}$, which bring $\mathbf{w}_0$ closer to the minimum. Computation of the increments $\Delta\mathbf{w}$ is performed by solving the following linear system

$$H \; \Delta\mathbf{w} \; = \; -\mathbf{F} \tag{6}$$

where vector $\mathbf{F}$ contains all the $\mathbf{f}_i^l$ vectors defined in Eq.(6). and the Hessian matrix H is defined as follows

$$H(m,n) = \frac{\partial F(m)}{\partial w(n)} \tag{7}$$

with $F(m)$ and $w(m)$ denoting the mth and nth element of the $\mathbf{F}$ and $\mathbf{w}$ vectors respectively. The Newton's method is attractive because it converges quadratically, given that the parameter vector is sufficiently close to a minimum of Eq.(1). However it may not converge if the initial estimates of the parameters are greatly in error. The method requires that the Hessian matrix H is positive definite, which may not be true even close to a minimum of Eq.(1) . Various modifications of the algorithm have been developed for its implementation [5]. Other techniques approximate the Hessian matrix by a simpler one, which does not require the computation of the second derivative of the error function. A well known approximation due to Levenberg and Marquardt [6], is the following

$$H = \mathbf{F}\mathbf{F}^T + \lambda\Omega \tag{8}$$

where λ is a positive factor, which controls the performance of the method and Ω an appropriately chosen matrix.

It should be emphasized at this point that the main problem in solving the system in Eq.(6) is its huge dimensionality, which is equal to

$$np = \sum_{l=1}^{L} N_{l-1} N_l \tag{9}$$

Furthermore neural networks provide the means for massive parallel computing, thus the above system of equations should be solved in a distributed simple parallel form. A simple way to do this is to let λ in (9) be a very large number and Ω be the identity matrix, so that matrix H becomes diagonal, with elements equal to λ. As a consequence, each single parameter in $\mathbf{w_0}$ can be updated independently of the others using Eq.(6). The resulting technique, which is the well known steepest descent method, is the basis for the derivation of the back-propagation learning algorithm . This algorithm has attracted much attention due to its computational simplicity, which is $O(np)$ operations per iteration and its suitability for parallel updating of all the weights of the network , or at least those of the same layer, during each iteration. Many successful applications of it have also been developed [7]. However this technique has many drawbacks. By using the above-mentioned approximation of H, it specifies only the direction of the required corrections to $\mathbf{w_0}$ and not the size of them. The method converges linearly and it may be inefficient, particularly as the minimum is approached. A way to improve the performance of the method is to use the overrelaxation tecnique, which is sometimes called momentum technique [2]. However most of the problems still remain.

For this reason the application of the standard quasi-Newton algorithm has been examined as a solution to the learning problem [3], but no efficient implementation in the form of an artificial neural network has been derived. Application of such methods in the solution of Eqs.(6) requires in general $O[(np)^2]$ operations per iteration. An approach has been recently developed [4], which implements Newton's method as a second order backpropagation algorithm. This approach uses an approximation of matrix H at iteration k, providing an iterative scheme of complexity $O(np)$. However this approximation is inefficient for real life applications and is useful only in cases of uncorrelated input signals. In this paper we follow a different approach. This approach permits the derivation of a new neural network learning algorithm, which posesses quadratic convergence close to a minimum, where it approximates the Newton method, and which can converge even if the initial estimates are relatively poor, as does the method of steepest descent.

III. The Least Squares Learning Algorithm

The main idea followed in this paper in order to derive an algorithm, which can be efficiently applied to distributed neural network training, is to let matrix Ω have the following block diagonal form

$$\Omega = \begin{pmatrix} \mathbf{f}_1^1(\mathbf{f}_1^1)^T & \dots & \mathbf{0} & \dots & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \dots & \mathbf{f}_{N_1}^1(\mathbf{f}_{N_1}^1)^T & \dots & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \dots & \mathbf{f}_1^L(\mathbf{f}_1^L)^T & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \dots & \mathbf{0} & \dots & \mathbf{f}_{N_L}^L(\mathbf{f}_{N_L}^L)^T \end{pmatrix} \tag{10}$$

This form contains the diagonal elements of $\mathbf{F}\mathbf{F}^T$, as well as the rest of the $\mathbf{f}_i^l(\mathbf{f}_i^l)^T$ elements around its diagonal ones, keeping thus second order information for the input parameters $\mathbf{w}_i^l$ of each neuron in the network. By letting λ be very large, matrix H is written in terms of Eq.(10)

$$H = \lambda\Omega \tag{11}$$

and due to the special form of matrix Ω, the system of equations (6) in the case of th ith neuron of the lth layer, for example, is written as follows

$$\lambda H_i^l \Delta\mathbf{w}_i^l = -\mathbf{f}_i^l \tag{12}$$

where

$$H_i^l = \sum_{r=1}^{K}\sum_{m=1}^{M} [\frac{\partial y(m)}{\partial \mathbf{w}_i^l}]_r [\frac{\partial y(m)}{\partial \mathbf{w}_i^l}]_r^T \tag{13}$$

As a result each parameter vector $\Delta\mathbf{w}_i^l$ can be updated independently of all others during each iteration in a distributed parallel form. The Marquardt-Levenberg or the Gauss-Newton least squares techniques [5]-[6] can be used for the solution of each Eq.(12).

In the neural network formulation the system of equations (12) implies that each neuron in the network tries to minimize the error function with respect to its own input connection weights $\mathbf{w}_i^l$ and that all the neurons of the network, or at least these of the same layer, may update their input weights during each iteration independently of all others. The damping factor $\mu = 1/\lambda$ prevents successive parameter estimates from oscillating. The overall system resembles the back-propagation type of solution, in that each neuron updates its input parameters independently and in that a damping factor is used to control convergence. However the fact that each neuron keeps second order information of its input parameter space, permits the proposed learning algorithm to converge much faster, when approaching a minimum of the error function. An example is presented in this paper, which verifies this statement. An adaptive choice of μ is also derived by letting it vary with neuron and iteration, which can be very useful in real life applications.

The learning algorithm has the distributed form of the backpropagation type of algorithms. It performs two sweeps through all the weights of the network at each iteration of it. On the first sweep an input vector is presented to the network. The input signals are propagated forward, so that each hidden and output neuron computes its own output. On the second sweep the error signals are computed at the top of the network as the difference between the desired and actual outputs. This information is then propagated down to the bottom layer and each neuron updates its input parameters using Eqs. (12)-(13).

The sequential version of the proposed iterative learning algorithm, when updating the parameter vector $\mathbf{w}_i^l$, is the following [6],[8]

$$\mathbf{w}_i^l(r) = \mathbf{w}_i^l(r-1) + P_i^l(r)\ e_i^l(r)\ (\mathbf{x}^{l-1})_r \tag{14}$$

$$P_i^l(r) = P_i^l(r-1) - \frac{1}{d_i^l}\ P_i^l(r-1)\ (\mathbf{x}^{l-1})_r (\mathbf{x}^{l-1})_r^T\ P_i^l(r-1) \tag{15}$$

where the scalars d and e are defined as follows

$$d_i^l = (b_i^l)^{-1} + (\mathbf{x}^{l-1})_r^T\ P_i^l(r-1)\ (\mathbf{x}^{l-1})_r \tag{16}$$

and

$$e_i^l(r) = \varepsilon_i^l - b_i^l\ (\mathbf{x}^l)_r^T [\mathbf{w}_i^l(r-1) - \mathbf{w}_i^l(0)] \tag{17}$$

Matrix P_i^l denotes the inverse of matrix H_i^l in Eq.(13). During each iteration of the algorithm, all the input data are processed sequentially one after the other and in the end the current parameter vector estimate replaces the initial vector $\mathbf{w}_i^l(0)$, so that the next iteration can begin. The procedure stops when the changes in all parameters are less than a very small threshold. Random values are initially chosen for $\mathbf{w}_i^l(0)$, while at the beginning of each iteration matrix $P_i^l(0)$ is chosen diagonal with relatively large values.

The computational complexity of the above algorithm in the case of an L layer network with N_l neurons in layer l is $O(np')$, where

$$np' = \sum_{l=1}^{L} N_l N_{l-1}^2 \tag{18}$$

which is greater than the corresponding backpropagation one, but less than the $O[(np)^2]$ computations , which would be required to solve the system of Eqs.(6) in its general form. Furthermore, the computational complexity can be further reduced to $O[(np)]$, particularly in signal processing applications, if fast implementations of the least squares algorithm are used [8].

IV. Adaptive Training of Multilayer Neural Networks

The algorithm presented in Eqs.(14)-(17) can be used to update the input parameter vector $\mathbf{w}_i^l$ of every neuron in the network by letting $l = 1, L$ and $i = 1, N_l$ as long as the quantities b_i^l and ε_i^l are computed appropriately. Since the updating of the input parameter vectors of all neurons takes place in the second sweep of the algorithm, as has been described earlier, appropriate formulas which update b_i^l and ε_i^l recursively from the top to the bottom of the network should be derived. The formulas for the ε_i^l are the same as the ones derived for the backpropagation algorithm [2]

$$\varepsilon_i^{l-1} = c_i^{l-1} \sum_{m=1}^{N_l} \varepsilon_m^l\ w_{im}^l \tag{19}$$

for $l = L-1, L-2 \cdots 2$. A similar, but more elaborate, recursive way of computing the non-negative quantities b_i^l in (16)-(17) can be derived [8]. A simple and effective approximate formula is the following

$$b_i^{l-1} = (c_i^{l-1})^2 \sum_{m=1}^{N_l} (w_{im}^l)^2 \, b_m^l \tag{20}$$

which has the same computational complexity as Eq.(19). The above equations together with Eqs.(14)-(15) form the proposed learning algorithm. It should however be added that a small positive damping factor, as was mentioned earlier, should be used to ensure convergence of the whole approach. The best value of μ to use can theoretically be defined by setting the derivatives of the error function in Eq.(1), with respect to μ, equal to zero, but the resulting equation is generally complex to solve in practice. By using a Taylor series approximation around a small value of μ and by definining the following quantities

$$R_i^l(K) = [\sum_{r=1}^{N} e_i^l(\mathbf{x}^{l-1})_r^T] P_i^l(K) [\sum_{r=1}^{N} e_i^l(\mathbf{x}^{l-1})_r] \tag{21}$$

it can be shown [8], that the reduction of the error function after each iteration is proportional to the sum of the scalars $R_i^l(K)$, computed during each iteration of the algorithm using Eqs.(14)-(17).

It should however be mentioned that different neurons will generally give different values of the $R_i^l(K)$ quantities and consequently different reductions of the error function. Using this fact we propose next an adaptive form of the algorithm, in which the damping factor μ varies with iteration and neuron . Such a choice, denoted by μ_i^l, which ensures a further reduction of the error in each iteration is [8]

$$\mu_i^l = \mu \frac{N_l \, R_i^l(K)}{\sum_{i=1}^{N_l} R_i^l(K)} \tag{22}$$

The proposed scheme uses Eqs. (14)-(17) and the same convergence factor for all the neurons of the network, when updating each parameter vector in Eq.(14). At the end of each iteration, however, i.e. after K input presentations, Eq. (22) is used for an additional distributed updating of the computed parameter vectors $\mathbf{w}_i^l$, which are then used by the next iteration of the algorithm.

V. An Example: The XOR Problem

This classical learning problem requires the use of at least one hidden layer in the network. The simplest form of such a network has two inputs, one output and comprises three neurons [2]. Four pairs of input (0,0),(0,1),(1,0),(1,1), and corresponding desired output 0,1,1,0 patterns are used for its training. A sequence of 20 patterns was generated and used as input data for each iteration of the learning algorithms. Ten different random choices of initial weights were generated in the range (-0.3,0.3) by feeding a random number generator with different initial seeds. The backpropagation-momentum algorithm was examined in the above cases with the overrelaxation term $a = 0.9$ and the convergence rate factor (a) $\mu = 0.1$ (b) $\mu = 0.3$ (c) $\mu = 0.5$. The performance of the proposed algorithm was also examined with the same values of μ and using an initial choice of $P_i^l(0) = 100$. Figures 1 and 2 illustrate the performance and speed of convergence of both algorithms, by showing the value of the error function in Eq.(1) in dB, in terms of the number of iterations.

The performance of the backpropagation algorithm with $\mu = 0.75$ is also shown in Figure 1. The adaptive choice of μ was then included in the proposed least squares algorithm and the result in the case of $\mu = 0.5$ is shown in both Figures for comparison purposes. The convergence speed of the momentum technique greatly varied with the choice of the initial weights. The respective forms of the proposed algorithm outperformed the momentum ones in all cases, being also less sensitive to initial conditions. The performance of the adaptive version was the best in all cases, being insensitive to the choice of the initial weights.

VI. Conclusions

A technique has been developed in this paper for the training of artificial neural networks, using a modification of the Marquardt-Levenberg optimization technique. An adaptive choice of the convergence rate factor μ, based on the contribution of each neuron in the minimization of the error function, was presented, which can be very useful in handling the problem of local minima of the error function. The proposed algorithm is more powerful, but also more elaborate than backpropagation. Moreover, it can be shown [8], that in some applications its computational complexity can be made similar to the backpropagation one, by using fast implementations of the least squares method.

References

[1] D.Parker,"Learning Logic", MIT Tech. Report TR-47, Center for Computational Research in Economics and Management Sciences, 1985.

[2] D.Rumelhart,G.Hinton,G.Williams,"Learning Internal Representations by Error Propagation", in "Parallel Distributed Processing", vol.1, eds. D.Rumelhart,J.McCleland, MIT Press, Cambridge MA, 1986.

[3]R.Watrous,"Learning Algorithms for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization", Tech.Report MS-CIS-87-51, LINC LAB 72, Univ. of Pennsylvania, 1986.

[4]D.Parker,"Second Order BackPropagation: Implementing an Optimal $O(n)$ Approximation to Newton's Method as an Artificial Neural Network" , submitted to Computer, 1987.

[5]D.Luenberger,"Linear and Nonlinear Programming", Reading MA, Addison Wesley, 1984.

[6]J.Beck,K.Arnold,"Parameter Estimation in Engineering and Science", J.Wiley and Sons, NY, 1976.

[7]T.Sejnowski,C.Rosenberg,"NET Talk: A Parallel Network That Learns to Read Aloud", Computer Systems, vol.1, pp. 145-168, 1987.

[8]S.Kollias, D.Anastassiou,"An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks", submitted to IEEE Transactions on Circuits and Systems, 1988.

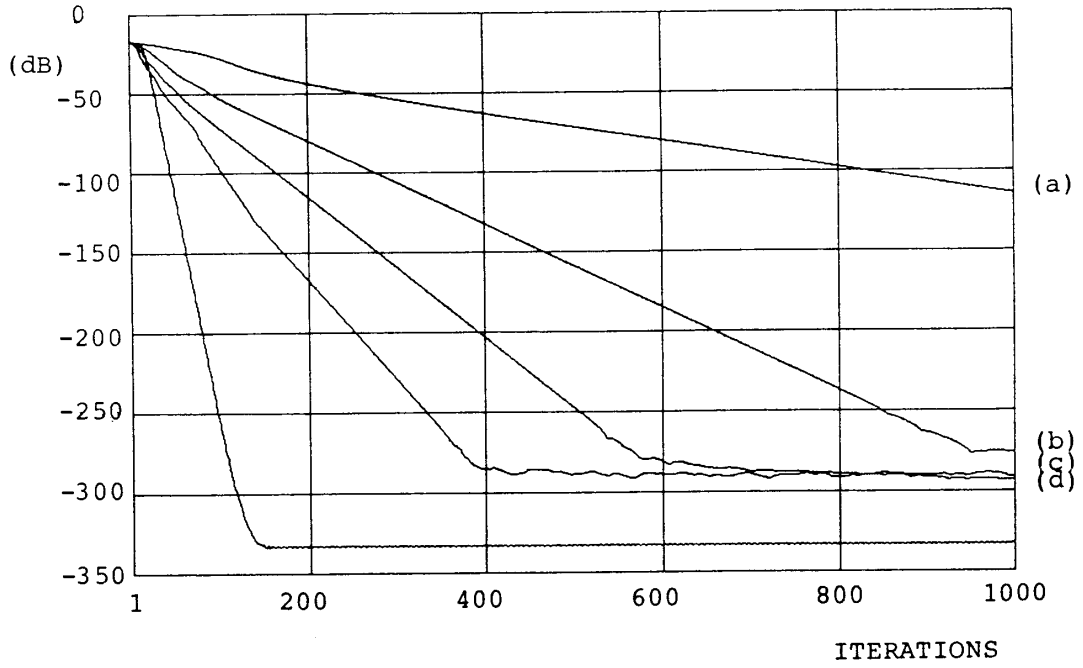

Fig.1 Average Performance of the backpropagation algorithm with four choices of the convergence rate factor, compared to the adaptive version of the proposed least squares algorithm.

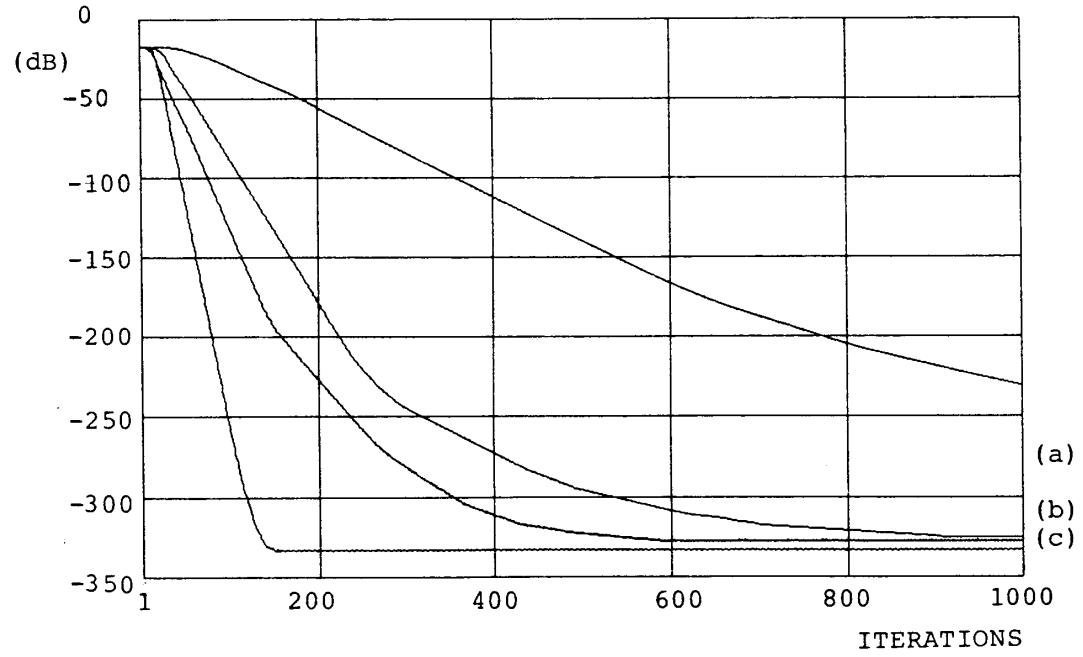

Fig.2 Average Performance of the iterative least squares algorithm with three choices of the convergence rate factor, compared to the adaptive version of it.