Letters

A Fast Feedforward Training Algorithm Using a Modified Form of the Standard Backpropagation Algorithm

S. Abid, F. Fnaiech, and M. Najim

***Abstract*—In this letter, a new approach for the learning process of multilayer feedforward neural network is introduced. This approach minimizes a modified form of the criterion used in the standard backpropagation algorithm. This criterion is based on the sum of the linear and the nonlinear quadratic errors of the output neuron. The quadratic linear error signal is appropriately weighted. The choice of the weighted design parameter is evaluated via rank convergence series analysis and asymptotic constant error values. The new proposed modified standard backpropagation algorithm (MBP) is first derived on a single neuron-based net and then extended to a general feedforward neural network. Simulation results of the 4-b parity checker and the circle in the square problem confirm that the performance of the MBP algorithm exceed the standard backpropagation (SBP) in the reduction of the total number of iterations and in the learning time.**

***Index Terms*—Linear quadratic error, modified standard backpropagation algorithm (MBP), neural network (NN), nonlinear quadratic error, standard backpropagation (SBP).**

I. Introduction

Neural networks, viewed as adaptive nonlinear filters or nonlinear systems such as nonlinear autoregressive moving average with exogenous input (NARMAX) models, have drawn great interest [9], [12], [17]. The traditional method for training a multilayer perceptron is the standard backpropagation (SBP) algorithm. Although it is successfully used in many real-world applications, the SBP algorithm suffers from a number of shortcomings. One of which is the rate at which the algorithm converges. Several iterations are required to train a small network, even for a simple problem. Moreover, the number of iterations needed to train a neural network depends heavily on other parameters namely the learning rate and the momentum. Reducing the number of iterations and speeding the learning time of NN are recent subjects of research [2]–[5], [8], [10], [15], [18]. In [10], the traditional Kalman filter method has been proposed to train neural networks and better results in terms of the reduction of the iteration number have been realized compared with the conventional SBP algorithm. To work with a system of linear equations, the authors used an inversion of the output layer nonlinearity and an estimation of the desired output summation in the hidden layers. In contrast with nonlinearity inversion, some authors have used other approaches as proposed in [3]–[5]. They apply the recursive least square (RLS) algorithm at each layer using the standard threshold logic type nonlinearity as an approximation of the sigmoid. These approaches yield fast training with respect to SBP algorithm but still are approximation dependant. To design fast training algorithms, which do not depend on the nonlinearity approximation, few attempts have been made [8], [15]. This paper may be considered among this set of attempts. Therefore, a new alternative algorithm, which is considerably faster than the SBP algorithm, is proposed. This new algorithm uses a modified form of the conventional SBP algorithm. It consists of minimizing the sum of the squares of linear and nonlinear errors for all output units and for the current pattern. The quadratic linear error is weighted by a coefficient λ. A design formula is derived, based on the series convergence rank theory. Improper choice of this design parameter may cause undesirable convergence behavior. To find the linear output error, we calculate the desired output summation by inverting the output layer nonlinearity. By deriving the optimization criterion with respect to the hidden learning coefficients, an estimation of the linear and nonlinear hidden errors can be determined. These estimates, along with the input vectors to the respective nodes, are used to produce an update set of weights using the modified form of the SBP algorithm. Training patterns are run through the network until convergence is reached. This letter is divided into five parts. Section II briefly presents the SBP algorithm. Section III introduces the new algorithm including the modifications of the SBP algorithm using the linear error in the updating equations. In Section IV, experimental results and comparisons between the two algorithms are given. Finally, in Section V, we present the main conclusions.

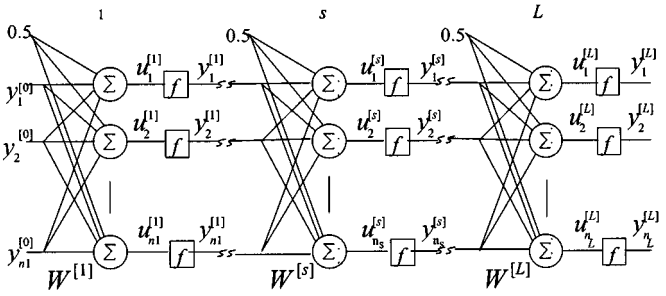

Fig. 1. Fully connected feedforward multilayer perceptron.

II. Review of the Standard Backpropagation Algorithm

The SBP algorithm has become the standard algorithm used for training multilayer perceptron as shown in Fig. 1. It is a generalized least mean squared (LMS) algorithm that minimizes a criterion equals to the sum of the squares of the errors between the actual and the desired outputs. This criterion is

$$E_p = \sum_{j=1}^{n_L} \left(e_{1j}^{[L]}\right)^2 \tag{1}$$

where the nonlinear error signal is

$$e_{1j}^{[L]} = d_j^{[L]} - y_j^{[L]}. \tag{2}$$

$d_j^{[L]}$ and $y_j^{[L]}$ are, respectively, the desired and the current outputs for the jth unit. P denotes in (1) the pth pattern, n_L is the number of the output units. The gradient descent method is given by

$$\Delta w_{ji}^{[s]} = -\mu \frac{\partial E_p}{\partial w_{ji}^{[s]}} \tag{3}$$

where $w_{ji}^{[s]}$ is the weight of the ith unit in the $(s-1)$th layer to the jth unit in the sth layer. Since the SBP algorithm is treated in the literature [9], [17], we summarized it by the following steps.

Manuscript received June 6, 2000; revised October 24, 2000. This work was supported by French/Tunisian Project CMCU(00F1125). An early condensed version of this work was presented in IEEE EURASIP Workshop (NSIP'99), Antalya, Turkey.

S. Abid and F. Fnaiech are with Laboratoire CEREP (CEntre de REcherche en Productique) E.S.S.T.T, 1008 Tunis, Tunisia (e-mail: Farhat.Fnaiech@esstt.rnu.tn).

M. Najim is with Equipe Signal et Image, Domaine Universitaire, BP 99, 33402 Talence Cedex, France (e-mail: najim@goelette.tsi.u-bordeaux.fr).

Publisher Item Identifier S 1045-9227(01)02049-5.

- Compute the error signals for the output layer from

$$e_j^{[L]} = f'\left(u_j^{[L]}\right) e_{1_j}^{[L]}. \tag{4}$$

- Compute the error signals for the hidden layers, i.e., for $s = L - 1$ to one, from

$$e_j^{[s]} = f'\left(u_j^{[s]}\right) \sum_{r=1}^{n_{s+1}} \left(e_r^{[s+1]} w_{rj}^{[s+1]}\right). \tag{5}$$

- Update the weights according to the following equation:

$$w_{ji}^{[s]}(k+1) = w_{ji}^{[s]}(k) + \mu e_j^{[s]} y_i^{[s-1]} \tag{6}$$

where μ is the learning coefficient and f' is the first derivative of f.

As we note from (2) and (3), the SBP algorithm minimizes the error at the output of the nonlinearity with respect to the weights.

In order to increase the convergence speed of the SBP, we propose to use a new form of the signal error based on the linear and nonlinear error delivered on each processing unit.

III. The Modified Standard Backpropagation Algorithm

A. New Learning Algorithm Used for Training a Single Layer Perceptron

First, let us develop the new algorithm for a neuron j located in any given layer s of the network. For the chosen pattern, we assume that for this given neuron the desired nonlinear output is known. Then the desired summation signal is directly calculated by inverting the nonlinearity.

The linear and the nonlinear current outputs are, respectively, given by

$$u_j^{[s]} = \sum_{i=0}^{n_{s-1}} w_{ji}^{[s]} y_i^{[s-1]} \tag{7}$$

$$f\left(u_j^{[s]}\right) = \frac{1}{1 + e^{-u_j^{[s]}}} = y_j^{[s]}. \tag{8}$$

Note that we have $n_s + 1$ inputs to the jth neuron. The nonlinear and linear errors are, respectively, equal to

$$e_{1_j}^{[s]} = d_j^{[s]} - y_j^{[s]} \tag{9}$$

$$e_{2_j}^{[s]} = ld_j^{[s]} - u_j^{[s]} \tag{10}$$

where $ld_j^{[s]}$ is given by

$$ld_j^{[s]} = f^{-1}\left(d_j^{[s]}\right). \tag{11}$$

Then, let us define the new optimization criterion E_p for the jth neuron and for the current pattern P

$$E_p \triangleq \tfrac{1}{2}\left(e_{1_j}^{[s]}\right)^2 + \tfrac{1}{2}\lambda\left(e_{2_j}^{[s]}\right)^2 \tag{12}$$

where λ is a weighting coefficient chosen so that (A10) holds (see the Appendix).

Hence, the weight update rule can be derived by just applying the gradient descent method to E_p, we get

$$\begin{aligned}\Delta w_{ji}^{[s]} &= -\mu \frac{\partial E_p}{\partial w_{ji}^{[s]}} = \mu\left[e_{1_j}^{[s]} \frac{\partial y_j^{[s]}}{\partial w_{ji}^{[s]}} + \lambda e_{2_j}^{[s]} \frac{\partial u_j^{[s]}}{\partial w_{ji}^{[s]}}\right] \\ &= \mu\left[e_{1_j}^{[s]} f'\left(u_j^{[s]}\right) y_i^{[s-1]} + \lambda e_{2_j}^{[s]} y_i^{[s-1]}\right] \\ &= \mu f'\left(u_j^{[s]}\right) e_{1_j}^{[s]} y_i^{[s-1]} + \mu\lambda e_{2_j}^{[s]} y_i^{[s-1]} \\ &= \mu e_j^{[s]} y_i^{[s-1]} + \mu\lambda e_{2_j}^{[s]} y_i^{[s-1]}. \end{aligned} \tag{13}$$

Compared with the SBP updating equations this new algorithm differs only by the term $\mu\lambda e_{2_j}^{[s]} y_i^{[s-1]}$. Specifically, for the purpose of illustration Fig. 2 is a synoptic bloc diagram of the new adaptive algorithm used to train a single neuron network.

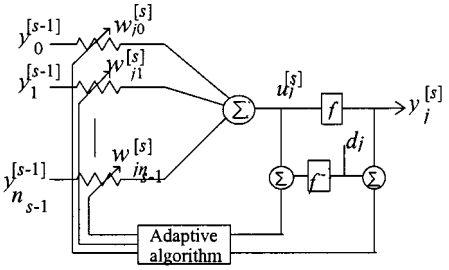

Fig. 2. Implementation of the new MBP algorithm for a single neuron j in a layer (s) of the NN.

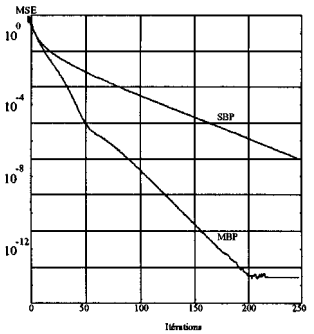

Fig. 3. Learning curve for the MBP algorithm versus SBP algorithm for the 5 input maximum detector $(\lambda = 0.01)$.

We have proved in the Appendix that this algorithm converges faster than the SBP algorithm for the chosen activation function and for an appropriate choice of λ. In fact the choice of λ is tricky problem. Whereas a very small value does not influence the convergence speed, a high value may cause the divergence of the algorithm. Note that when $\lambda = 0$, we get the SBP algorithm.

B. Practical Example For a Single Neuron Perceptron: Five Input Maximum Detector

This application aims at finding the maximum of five input values. If the first input is maximum (1, 0, 0, 0, 0), the output is equal to (0.1). If the second input is maximum (0, 1, 0, 0, 0), the output is equal to $(0.1 + 0.2)$, and so on. From Fig. 3 it is clear that the new algorithm converges faster than the SBP for a suitable choice of λ.

C. New Algorithm Extended for the Multilayer Perceptron

So far, we applied the new algorithm to a single-layer perceptron. Let us extend it to the multilayer perceptron.

Using the new optimization criterion given by

$$E_p = \sum_{j=1}^{n_L} \tfrac{1}{2}\left(e_{1_j}^{[L]}\right)^2 + \sum_{j=1}^{n_L} \tfrac{1}{2}\lambda\left(e_{2_j}^{[L]}\right)^2 \tag{14}$$

we shall derive the updating equations for the output and hidden layers.

1) Learning of the Output Layer: In the output layer both linear and nonlinear errors are known, the application of the gradient descent method to E_p leads to

$$\begin{aligned}\Delta w_{ji}^{[L]} &= -\mu \frac{\partial E_p}{\partial w_{ji}^{[L]}} = \mu e_{1_j}^{[L]} \frac{\partial y_j^{[L]}}{w_{ji}^{[L]}} + \mu\lambda e_{2_j}^{[L]} \frac{\partial u_j^{[L]}}{\partial w_{ji}^{[L]}} \\ &= \mu e_{1_j}^{[L]} \frac{\partial y_j^{[L]}}{\partial u_j^{[L]}} \frac{\partial u_j^{[L]}}{\partial w_{ji}^{[L]}} + \mu\lambda e_{2_j}^{[L]} y_i^{[L-1]} \end{aligned} \tag{15}$$

$$\Delta w_{ji}^{[L]}(k) = \mu f'\left(u_j^{[L]}\right) e_{1_j}^{[L]} y_i^{[L-1]} + \mu\lambda e_{2_j}^{[L]} y_i^{[L-1]}. \tag{16}$$

TABLE I
THE NEW MODIFIED BACKPROPAGATION ALGORITHM

Step 1: Initialization:
* From layer s=1 to L, set all $y_0^{[s-1]}$ to values different from 0, (0.5 for example).
* Randomize all the weights $w_{ji}^{[s]}$ at random values.
* Choose a small value λ.

Step 2: Select training pattern:
Select an input/output pattern to be processed into the network.

Step 3: Run selected pattern p through the network for each layer (s), (s = 1... L) and calculate for each node j the:
- Summation outputs: equation (7)
- The nonlinear outputs: equation (8)

Step 4: Error signals:
* For the output layer L
- Calculate the desired summations : equation (11) for s=L
- Calculate the output errors: equation (2)
- The nonlinear output errors: equation (10) for $s = L$
* For the hidden layers : s =L-1 to 1
- Evaluate the nonlinear estimation errors: equation (23)
- Evaluate the linear estimation errors: equation (24)

Step 5: Updating the synaptic coefficients:
For any node j of the layer s=1 to L modify the synaptic coefficients using equation (22)

Step 6: Testing for the ending of the running:
Various criteria are tested for ending. We can use the mean square error of the network output as a convergence test or we can run the program for a fixed number of iterations. If the condition is not verified, go back to Step 2.

In the Appendix, it is shown that the convergence of this algorithm is faster than the SBP.

2) Learning of the Hidden Layers: In the hidden layers, both the linear and the nonlinear outputs are unknown and must be then estimated. First, let us apply the gradient descent method to E_p for the layer $[L-1]$ and then generalize the results for the other hidden layers. Consider the following relationship:

$$\begin{aligned}\Delta w_{ir}^{[L-1]} &= -\mu \frac{\partial E_p}{\partial w_{ir}^{[L-1]}} \\ &= \mu \sum_{j=1}^{n_L} e_{1_j}^{[L]} \frac{\partial y_j^{[L]}}{\partial u_j^{[L]}} \frac{\partial u_j^{[L]}}{\partial y_i^{[L-1]}} \frac{\partial y_i^{[L-1]}}{\partial u_i^{[L-1]}} \frac{\partial u_i^{[L-1]}}{\partial w_{ir}^{[L-1]}} \\ &\quad + \mu\lambda \sum_{j=1}^{n_L} e_{2_j}^{[L]} \frac{\partial u_j^{[L]}}{\partial y_i^{[L-1]}} \frac{\partial y_i^{[L-1]}}{\partial u_i^{[L-1]}} \frac{\partial u_i^{[L-1]}}{\partial w_{ir}^{[L-1]}} \end{aligned} \tag{17}$$

which yields

$$\begin{aligned}\Delta w_{ir}^{[L-1]} &= \mu y_r^{[L-2]} f'\left(u_i^{[L-1]}\right) \left[\sum_{j=1}^{n_L} e_{1_j}^{[L]} f'\left(u_j^{[L]}\right) w_{ji}^{[L]}\right] \\ &\quad + \mu\lambda y_r^{[L-2]} f'\left(u_i^{[L-1]}\right) \left[\sum_{j=1}^{n_L} e_{2_j}^{[L]} w_{ji}^{[L]}\right]. \end{aligned} \tag{18}$$

Referring to the updating equation (16) for the output layer, it can be assumed that the term

$$\sum_{j=1}^{n_L} e_{1_j}^{[L]} f'\left(u_j^{[L]}\right) w_{ji}^{[L]}$$

is an estimated nonlinear error and the term

$$f'\left(u_i^{[L-1]}\right) \sum_{j=1}^{n_L} e_{2_j}^{[L]} w_{ji}^{[L]}$$

is an estimated linear error. Furthermore, we define the estimated nonlinear error in the hidden layer $[L-1]$ for the ith neuron by

$$e_{1_i}^{[L-1]} = \sum_{j=1}^{n_L} e_{1_j}^{[L]} f'\left(u_j^{[L]}\right) w_{ji}^{[L]} \tag{19}$$

and the estimated linear error by

$$e_{2_i}^{[L-1]} = f'\left(u_i^{[L-1]}\right) \sum_{j=1}^{n_L} e_{2_j}^{[L]} w_{ji}^{[L]}. \tag{20}$$

The updating equation for the $[L-1]$th layer takes the same form of the one obtained for the output layer

$$\Delta w_{ir}^{[L-1]} = \mu y_r^{[L-2]} f'\left(u_i^{[L-1]}\right) e_{1_i}^{[L-1]} + \mu\lambda y_r^{[L-2]} e_{2_i}^{[L-1]}. \tag{21}$$

The procedure of derivation may be performed layer by layer. Hence, for a given layer s the updating equation (21) becomes

$$\Delta w_{ji}^{[s]} = \mu y_i^{[s-1]} f'\left(u_j^{[s]}\right) e_{1_j}^{[s]} + \mu\lambda y_i^{[s-1]} e_{2_j}^{[s]} \tag{22}$$

where

$$e_{1_j}^{[s]} = \sum_{r=1}^{n_{s+1}} e_{1_r}^{[s+1]} f'\left(u_r^{[s+1]}\right) w_{rj}^{[s+1]} \tag{23}$$

TABLE II
MULTIPLICATION OPERATION NUMBER OF THE MBP/SBP ALGORITHM FOR ONE P*ATTERN

Algorithm	SBP	MBP
Errors	$2n_L + \sum_{s=1}^{L-1} n_s(n_{s+1}+2)$	$4n_L + \sum_{s=1}^{L-1} n_s(n_{s+1}+2) + \sum_{s=1}^{L-1} n_s(n_{s-1}+1)$
Updating	$\sum_{s=1}^{L} n_s(n_{s-1}+2)$	$\sum_{s=1}^{L} n_s(n_{s-1}+2) + \sum_{s=1}^{L} n_s(n_{s-1}+2)$

and

$$e_{2j}^{[s]} = f'\left(u_j^{[s]}\right) \sum_{r=1}^{n_{s+1}} e_{2r}^{[s+1]} w_{rj}^{[s+1]}. \tag{24}$$

Finally, Table I summarizes the new algorithm.

D. Comparison of the Computation Complexity

Table II gives a comparison of the number of multiplication operations needed for each algorithm to compute the error signals and the updating equations for one pattern. Obviously, the proposed algorithm is slightly more complex than the SBP, while in the sequel, we will see that it has a faster convergence behavior than the SBP in number of training iterations and in computing time.

IV. EXPERIMENTAL RESULTS

To compare the performances of the new algorithm with respect to the conventional SBP, both algorithms are used to train the networks for the same problem. In this paper we present two examples, the 4-b parity checker (logic problem) [3], and the circle in the square problem (analog problem) [18].

For both algorithms, the learning parameters are selected with respect to a performance criterion. However, an exhaustive search for the best possible parameters is beyond the scope of this paper, and optimal values may exist for either algorithms. In order to make suitable comparison we kept the same NN size. In all applications, we have used a NN structure with eight hidden neurons and one output neuron. The input neuron number depends on the application: four for the 4-b parity checker and two for the circle in the square problem.

A. The 4-b Parity Checker

The aim of this application is to determine the parity of a 4-b binary number. The NN inputs are logic values (0.5 for the higher level; -0.5 for the lower level). In each iteration we present to the NN the 16 input combinations with their desired outputs (0.1 for the lower level and 0.9 for the higher level).

Fig. 4 shows the average of the MSE over 100 different weight initialization trials taken in the range of ± 5, versus the iteration number for both algorithms. It is clear that the MBP is highly faster than the SBP. Table III shows that the new algorithm remains below 10 e-4 after only 130 iterations as opposed to the SBP algorithm at 385 iterations. Based on this experiments, clearly we see that there is an improvement ratio, nearly three, for the number of iterations, and about 2.3 for the convergence time.

In [3], the authors used the recursive least square (RLS) algorithm in order to accelerate the learning speed and they used a NN with only four hidden nodes. A significant improvement in the learning speed was obtained. By using four hidden nodes as in [3], the MBP steel faster than the SBP and require almost the same iteration number than in [3] to reach a small threshold.

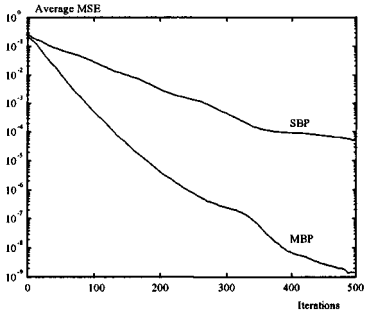

Fig. 4. Learning curve for the MBP versus SBP algorithm for the 4-b parity checker. ($\mu = 10$; $\lambda = 0.01$).

TABLE III
COMPARISON OF THE CPU TIME NEEDED FOR THE CONVERGENCE OF THE MBP/SBP

	4-byte parity checker problem (Threshold=10e-4)		
	Number of iterations	CPU Time (s)/iteration	Total time of convergence(s)
SBP	385	5.4210^{-3}	2.086
MBP	130	7.0110^{-3}	0.911

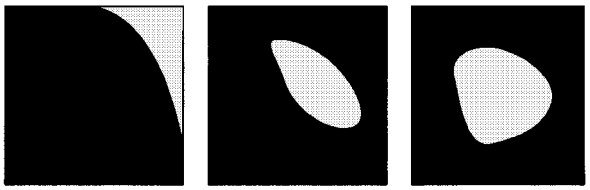

Fig. 5. SBP ($\mu = 0.5$); (a) Iterations 2. MSE = 0.193. (b) Iterations 30. MSE = 0.114. (c) Iterations 100. MSE = 0.0259.

B. Circle in the Square Problem

In this application, the NN have to decide if a point of coordinate (x, y) varying from -0.5 to $+0.5$ is in the circle of radius equals to 0.35. The input coordinates are selected randomly. If the distance of training point from the origin is less than 0.35, the desired output is assigned the value 0.1 indicating that the point is inside the circle. A distance greater than 0.35 means that the point is outside the circle and the desired output becomes 0.9. Training patterns are presented to the network alternating between the two classes (inside and outside the circle). In one iteration we present to the network 100 input–output patterns. The output of the network at various iteration numbers is shown in Fig. 5(a)–(c) for the SBP algorithm and Fig. 6(a)–(c) for the new algorithm. It is seen that the new algorithm forms a circle in ten to 30 iterations whereas the SBP algorithm needs more than 100 iterations.

It should be also noted that this method of specifying the desired output forces the "training surface" to assume a top hat-like shape, a

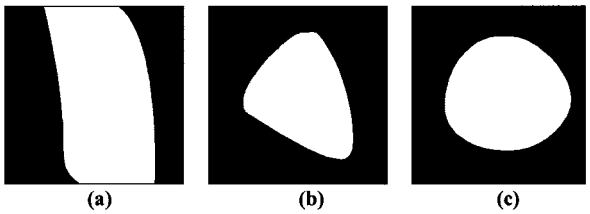

Fig. 6. MBP. ($\mu = 0.5$; $\lambda = 0.7$); (a) Iterations 2. MSE = 0.229. (b) Iterations 10. MSE = 0.0358. (c) Iterations 30. MSE = 0.0156.

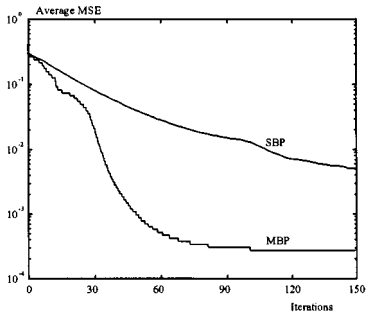

Fig. 7. Learning curve for the MBP versus SBP algorithm for the circle in the square problem. ($\mu = 0.5$; $\lambda_0 = 0.7$).

TABLE IV
COMPARISON OF THE CPU TIME NEEDED FOR THE CONVERGENCE OF THE MBP/SBP

	Circle in the square problem (Threshold=0.01)		
	Number of iterations	CPU Time (s)/iteration	Total time of convergence(s)
SBP	110	29.6610^{-2}	32.62
MBP	34	34.5410^{-2}	11.74

constant high value outside the circle and a constant low value inside the circle; which is not the best choice for the circle problem. Since during the training period, fewer errors occur when the output values are closer to 0.1 and 0.9, the algorithms tend to sacrifice the transition region in order to flatten the inside and outside regions. The result is less than perfect circle. A training surface with continuous first derivatives and gradual transition region would improve the appearance of the circle.

Fig. 7 shows the average of the MSE over 100 different weight initialization trials taken in the range of ± 3, versus the iteration number for both algorithms during training. Table IV shows that the new algorithm remains below 0.01 after 34 iterations as opposed to the SBP algorithm at 110 iterations. Notice that there is an improvement ratio of about 3.2 in the number of iterations and about 2.8 in the learning time. In [18] the authors used nine hidden nodes in this experience and they show that their new RLS algorithm has an improvement ratio about 2.35 with respect to the backpropagation algorithm with momentum.

V. CONCLUSION

In this paper, we have proposed a new fast algorithm for training neural networks based on a criterion taking into account the linear and nonlinear signal errors. The convergence of the new algorithm requires less iterations than the SBP. For a suitable choice of the learning parameters, the optimal range values of the weighting parameter λ are given in the Appendix. In some applications, it is necessary to decrease λ in order to speed up the convergence of the MBP algorithm. This can be explained by inspecting (A10). We shall remark that when $w_{ji}^{[s]}$ tends to $(w_{ji}^{[s]})^*$, $F'(w_{ji}^{[s]})$ tends to zero and the range in which lie the optimal values of λ becomes small during the learning process.

APPENDIX

For the neuron j located in the layer s of the network, we will assume that we know its desired nonlinear output for the chosen pattern. Then the desired summation can be directly computed by inverting the nonlinearity. The descent gradient method applied to E_p leads to

$$\begin{aligned}\frac{\partial E_p}{\partial w_{ji}^{[s]}} &= -e_{1_j}^{[s]}\frac{\partial y_j^{[s]}}{\partial w_{ji}^{[s]}} - \lambda e_{2_j}^{[s]}\frac{\partial u_j^{[s]}}{\partial w_{ji}^{[s]}} \\ &= -e_{1_j}^{[s]}\frac{\partial y_j^{[s]}}{\partial u_j^{[s]}}\frac{\partial u_j^{[s]}}{\partial w_{ji}^{[s]}} - \lambda e_{2_j}^{[s]}\frac{\partial u_j^{[s]}}{\partial w_{ji}^{[s]}} \\ &= -e_{1_j}^{[s]} f'\left(u_j^{[s]}\right) y_i^{[s-1]} - \lambda e_{2_j}^{[s]} y_i^{[s-1]}. \end{aligned} \tag{A1}$$

In the case of the SBP algorithm: $\lambda = 0$

$$w_{ji}^{[s]}(k+1) = w_{ji}^{[s]}(k) + \mu f'\left(u_j^{[s]}\right) e_{1_j}^{[s]} y_i^{[s-1]} \triangleq F_i\left(w_{ji}^{[s]}\right). \tag{A2}$$

For the new algorithm: $\lambda > 0$

$$\begin{aligned} w_{ji}^{[s]}(k+1) &= w_{ji}^{[s]}(k) + \mu f'\left(u_j^{[s]}\right) e_{1_j}^{[s]} y_i^{[s-1]} + \mu\lambda e_{2_j}^{[s]} y_i^{[s-1]} \\ &\triangleq H_i\left(w_{ji}^{[s]}\right). \end{aligned} \tag{A3}$$

Note that we have

$$H_i\left(w_{ji}^{[s]}\right) = F_i\left(w_{ji}^{[s]}\right) + \mu\lambda y_i^{[s-1]} e_{2_j}^{[s]}. \tag{A4}$$

Consider that both functions F_i and H_i admits a fixed point in a finite range of the real set $\Re$, and also assume that this point is unique for both functions. Hence, the updating equations (A2) and (A3) may be viewed as two series of functions and thus will generate two series of values noted $\{t_{ji} = F_i(w_{ji})\}$ and $\{h_{ji} = H_i(w_{ji})\}$, to converge to their fixed points assumed to be the same.

To compare the convergence speed of these two series we should compare their ranks and their asymptotic constants error values. We will use the following theorems and definition.

Theorem 1 [6], [11]: Let the series $\{x_k\}$ is generated by $x_{k+1} = F(x_k)$. If this series converges to (x) and if F is sufficiently differentiable in the proximity of (x) then the rank of the series $\{x_k\}$ is the smallest integer number r verifying

$$F'(x) = \cdots = F^{(r-1)}(x) = 0 \quad \text{and} \quad F^{(r)}(x) \neq 0.$$

We have also

$$\lim_{k\to\infty} \frac{x_{k+1} - x_k}{(x_k - x)^r} = \frac{F^{(r)}(x)}{r!}.$$

Definition: Following the same conditions of the previous theorem, we shall define the asymptotic error constant "c" of the series $\{x_k\}$ by

$$c \triangleq \left|\lim_{k\to\infty} \frac{x_{k+1} - x_k}{(x_k - x)^r}\right| = \left|\frac{F^{(r)}(x)}{r!}\right|.$$

Theorem 2 [6], [11]: Define $\{t_{ji}\}$ and $\{h_{ji}\}$ as two series with, respectively, ranks r and p. If $r > p$, $\{t_{ji}\}$ converges faster than $\{h_{ji}\}$.

If $r = p$ and if the asymptotic error constant of $\{t_{ji}\}$ is less than the one of $\{h_{ji}\}$, $\{t_{ji}\}$ converges faster than $\{h_{ji}\}$.

In order to compare the convergence speed of the series defined by (A2) and (A3), we shall compare their ranks. First note that both functions F_i and H_i are differentiable over $\Re$. We shall assume that they will converge to the same fixed point w_{ji}^*.

A. Evaluation of the Rank of $F_i(w_{ji}^{[s]})$

In order to evaluate the rank r_1 of $F_i(w_{ji}^{[s]})$, we shall compute the first derivative of F_i as

$$\begin{aligned}\frac{\partial F_i\left(w_{ji}^{[s]}\right)}{\partial w_{ji}^{[s]}} &= 1+\mu y_i^{[s-1]}\left[f'\left(u_j^{[s]}\right)\left(d_j^{[s]}-y_j^{[s]}\right)\right]' \\ &= 1+\mu y_i^{[s-1]}\left[\frac{\partial f'\left(u_j^{[s]}\right)}{\partial w_{ji}^{[s]}}e_{1j}^{[s]}-f'\left(u_j^{[s]}\right)\frac{\partial y_j^{[s]}}{\partial w_{ji}^{[s]}}\right] \end{aligned} \tag{A5}$$

which gives

$$\begin{aligned}\frac{\partial F_i\left(w_{ji}^{[s]}\right)}{\partial w_{ji}^{[s]}} = 1-\mu\left(y_i^{[s-1]}\right)^2 f'\left(u_j^{[s]}\right) \\ \cdot\left[f'\left(u_j^{[s]}\right)-\left(1-2f\left(u_j^{[s]}\right)\right)e_{1j}^{[s]}\right].\end{aligned} \tag{A6}$$

Case 1: If $y_i^{[s-1]} = 0$ or

$$\left[f'\left(u_j^{[s]}\right)-\left(1-2f\left(u_j^{[s]}\right)\right)e_{1j}^{[s]}\right]=0$$

$F_i'(w_{ji}^{[s]}) = 1$. Hence, it follows that $r_1 = 1$.

Case 2: If $y_i^{[s-1]} \neq 0$ and

$$\left[f'\left(u_j^{[s]}\right)-\left(1-2f\left(u_j^{[s]}\right)\right)e_{1j}^{[s]}\right]\neq 0$$

and by taking into account that $0 < f(u_j^{[s]}) < 1$, $0 < f'(u_j^{[s]}) < 1$ and $0 < (y_i^{[s-1]})^2 < 1$ we obtain

$$\begin{aligned}-1 < -\left(y_i^{[s-1]}\right)^2 f'\left(u_j^{[s]}\right) \\ \cdot\left[f'\left(u_j^{[s]}\right)-\left(1-2f\left(u_j^{[s]}\right)\right)e_{1j}^{[s]}\right]<1\end{aligned}$$

then $F_i'(w_{ji}^{[s]}) > 0$ while

$$\begin{aligned}0<\mu \\ <\frac{1}{\left(y_i^{[s-1]}\right)^2 f'\left(u_j^{[s]}\right)\left[f'\left(u_j^{[s]}\right)-\left(1-2f\left(u_j^{[s]}\right)\right)e_{1j}^{[s]}\right]}\end{aligned}$$

then $r_1 = 1$. For both cases notice that $F_i'(w_{ji}^{[s]}) > 0$, it results that the rank of this series is equal to one. The asymptotic constant of F_i is equal to

$$c_1 = \left|F_i'\left(w_{ji}^{[s]}\right)\right| = F_i'\left(w_{ji}^{[s]}\right). \tag{A7}$$

B. Evaluation of the Rank of $H_i(w_{ji}^{[s]})$

Taking the derivative of $H_i(w_{ji}^{[s]})$, yields

$$\frac{\partial H_i\left(w_{ji}^{[s]}\right)}{\partial w_{ji}^{[s]}} = \frac{\partial F_i\left(w_{ji}^{[s]}\right)}{\partial w_{ji}^{[s]}} - \mu\lambda y_i^{[s-1]}\frac{\partial u_j^{[s]}}{\partial w_{ji}^{[s]}} \tag{A8}$$

$$H_i'\left(w_{ji}^{[s]}\right) = F_i'\left(w_{ji}^{[s]}\right) - \mu\lambda y_i^{[s-1]}. \tag{A9}$$

Case 1: If $F_i'(w_{ji}^{[s]}) = \mu\lambda y_i^{[s-1]}$ i.e., $H_i'(w_{ji}^{[s]}) = 0$ then the rank of this series become higher or equal to two, i.e., $r_2 \geq 2$. We obtain a convergence speed which is at least twice of the one obtained by the SBP.

Case 2: If $F_i'(w_{ji}^{[s]}) \neq \mu\lambda y_i^{[s-1]}$ i.e., $H_i'(w_{ji}^{[s]}) \neq 0$ then $r_2 = 1$. The series H_i will converge faster than F_i if $c_2 < c_1$ i.e.,

$$\left|H_i'\left(w_{ji}^{[s]}\right)\right| < \left|F_i'\left(w_{ji}^{[s]}\right)\right|$$

(c_2 is the asymptotic constant error of H_i). This inequality always holds if

$$0<\lambda<\frac{2F_i'\left(w_{ji}^{[s]}\right)}{\mu\left(y_i^{[s-1]}\right)^2} \tag{A10}$$

we have assumed that $(y_i^{[s-1]})^2 \neq 0$. If $(y_i^{[s-1]})^2 = 0$, $H_i'(w_{ji}^{[s]}) = F_i'(w_{ji}^{[s]})$, and $c_2 = c_1$.

In this case the two series have a similar convergence behavior, but it should be noted here that this case could occur only in the input layer for null components vectors of the learning input pattern. For the hidden and output layers we always have $(y_i^{[s-1]})^2 \neq 0$ so the convergence is always faster than the SBP.

In some practical applications we find it important to take λ a decreasing parameter during the learning process (from one to zero).

ACKNOWLEDGMENT

The authors thank Prof. S. Gherissi, Supervisor of CEREP and the Dean of ESSTT, for his support of this work. The authors are also indebted to the reviewers for their helpful comments which improved the quality of this letter.

REFERENCES

[1] S. Abid, F. Fnaiech, and M. Najim, "Evaluation of the feedforward neural network covariance matrix error," in *IEEE Proc. Int. Conf. Acoust., Speech, Signal Processing (ICASSP'2000)*, Istanbul, Turkey, June 5–9.

[2] ——, "A new fast neural-network training algorithm based on a modified standard backpropagation (MBP) criterion optimization," in *Proc. IEEE-EURASIP Workshop Nonlinear Signal Image Processing (NSIP'99)*, Antalya, Turkey, June 20–23, 1999, pp. 733–737.

[3] I. M. R. Azimi-Sadjadi and R. J. Liou, "Fast learning process of multilayer neural network using recursive least squares method," *IEEE Trans. Signal Processing*, vol. 40, Feb. 1992.

[4] M. R. Azimi-Sadjadi and S. Citrin, "Fast learning process of multilayer neural nets using recursive least squares technique," in *Proc. IEEE Int. Conf. Neural Networks*, Washington, DC, May 1989.

[5] M. R. Azimi-Sadjadi, S. Citrin, and S. Sheedvash, "Supervised learning process of multilayer perceptron neural networks using fast recursive least squares," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing (ICASSP'90)*, NM, Apr. 1990, pp. 1381–1384.

[6] C. Berezinski, *Algorithmique Numérique*. Paris, France: Edition Marketing, 1988.

[7] I. S. Berezin and N. P. Zhidkov, *Computing Methods*. New York: Pergamon , 1965.

[8] V. N. B. Carayiannis and A. V. Venetsanopoulos, "Fast learning algorithm for neural networks," *IEEE Trans. Circuits Syst. II*, vol. 39, pp. 453–474, 1992.
[9] A. Cichocki and R. Unbehauen, *Neural Network for Optimization and Signal Processing*. Chichester, U.K.: Wiley, 1993.
[10] F. Fnaiech, D. Bastard, V. Buzenac, R. Settineri, and M. Najim, "A fast Kalman filter based new algorithm for training feedforward neural networks," in *Proc. EUSIPCO 94*, Edinburg, U.K., Sept. 13–16, 1994.
[11] D. Graupe, *Time Series Analysis, Identification Adaptive Filtering*. Malabar, FL: Krieger, 1984.
[12] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice-Hall, 1986.
[13] D. R. Hush and B. G. Horne, "Progress in supervised neural network," *IEEE Signal Processing Mag.*, vol. 10, no. 1, pp. 8–39, July 1993.
[14] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*. New York: Wiley, 1996.
[15] F. B. Konig and F. Barmann, "A learning algorithm for multilayered neural networks based on linear squares problems," *Neural Networks*, vol. 6, pp. 127–131, 1993.
[16] Y. Liguni, H. Sakai, and H. Tokumaru, "A real-time algorithm for multilayered neural network based on the extended Kalman filter," *IEEE Trans. Signal Processing*, vol. 40, pp. 959–966, Apr. 1992.
[17] R. P. Lippman, "An introduction to computing with neural networks," *IEEE ASSP. Mag.*, vol. 4, no. 2, Apr. 1987.
[18] R. S. Scalero and N. Tepedelenlioglu, "A fast new algorithm for training feedforward neural networks," *IEEE Trans. Signal Processing*, vol. 40, no. 1, pp. 202–210, Jan. 1992.
[19] R. J. Serfling, *Approximation Theorems of Mathematical Statistics*. New York: Wiley, 1980.
[20] V. R. Setino and L. C. K. Hui, "Use of a quasi-Newton method in feedforward neural network construction algorithm," *IEEE Trans. Neural Networks*, vol. 6, pp. 273–277, Jan. 1995.
[21] G. J. Wang and C. C. Chen, "A fast multilayer neural network training algorithm based on the layer-by-layer optimization procedures," *IEEE Trans. Neural Networks*, vol. 7, pp. 768–775, May 1996.

Feedforward Networks Training Speed Enhancement by Optimal Initialization of the Synaptic Coefficients

Jim Y. F. Yam and Tommy W. S. Chow

***Abstract*—This letter aims at determining the optimal bias and magnitude of initial weight vectors based on multidimensional geometry. This method ensures the outputs of neurons are in the active region and the range of the activation function is fully utilized. In this letter, very thorough simulations and comparative study were performed to validate the performance of the proposed method. The obtained results on five well-known benchmark problems demonstrate that the proposed method deliver consistent good results compared with other weight initialization methods.**

I. Introduction

Feedforward neural networks (NNs) are usually trained by the backpropagation (BP) algorithm, which is often too slow for most applications. One of the approaches to speed up the training rate is to estimate optimal initial weights [1]–[9]. In the followings, a brief summary on major methods determining initial weights is provided. Shepanski regards the training of a multilayer feedforward network as an optimal estimation problem [1]. An optimal set of weights is determined using the least squares criterion employing a standard pseudoinverse matrix technique. Masters uses a least squares method as a part of his weight initialization algorithm [2]. For a network with one hidden layer, he suggests that a simulated annealing method or a genetic algorithm should be used for initializing the weights between the input and the hidden layers. Yam and Chow also proposed two weight initialization methods based on a least squares approach [3], [4]. In [3], the system is assumed to be linear and the parameters relating the input and the output are obtained by a linear least squares method. From these parameters, the optimal initial weights between layers are successively determined by factorization. However, this method cannot be applied to NNs in which the number of hidden neurons is less than the number of neurons in the preceding layer plus one. In [4], the output of hidden neurons are assigned with values in the range of nonsaturation region, the optimal initial weights between the input and hidden layers are evaluated by a linear algebraic method. The actual outputs of the hidden neurons are obtained by propagating the input patterns through the networks. The optimal weights between the hidden layer and the output layer can then be evaluated by using the least squares method.

Nguyen and Widrow speed up the training process by setting the initial weights of the hidden layer so that each hidden node is assigned to approximate a portion of the range of the desired function at the beginning of training process [5]. By approximating the activation function with piece-wise linear segments, the weights are evaluated. Next, the thresholds of neurons are selected by assuming the input variables are kept between -1 and one. Osowski also approximates the activation function with piece-wise linear segments [6]. However, he suggests an explicit method to determine the number of hidden neurons, and he uses the information of the desired function $y = f(x)$ to determine the initial weights. The weights and the bias of each neuron are determined in such a way that this neuron enters its active (middle) region of the proper section. The weights to the output layer are finally evaluated using the change of y in the corresponding section with the bias of the output neuron set to zero. Jia proposed a weight initialization method ensuring that each neuron operates at the middle region of the activation function [13], but comparative performance with other methods is not available.

Drago and Ridella propose a method called statistically controlled activation weight initialization (SCAWI) to find the optimal initial weights [7]. The idea of the SCAWI method is to prevent neurons from saturation in the early stage through properly initializing the weights. Drago and Ridella determine the maximum magnitude of the weights by statistical analysis. They show that the maximum magnitude of the weights is a function of the paralyzed neuron percentage (PNP), which is in turn related to the convergence speed. By determining the optimal range of PNP through computer simulations, the maximum magnitude of the weights are obtained. Their results show that the time required to reach the defined convergence criterion is reduced. Wessels and Bernard also propose a method for evaluating the maximum magnitude of weights based on a statistical method. The method provides certain performance enhancement, but additional criteria are required to confine the selection of weights to avoid the solution being trapped in local minima [8].

Thimm and Fiesler examined various weight initialization methods for multilayer perceptrons. In their work, they applied various methods to eight real-world benchmark problems [9]. They also performed a large number of simulations for high-order networks using the same eight data sets. The results of these experiments were compared to weight initialization techniques for multilayer perceptrons. Their results show that a convenient weight initialization method for high-order perceptrons with the shifted/scaled logistic and the identity

Manuscript received July 5, 2000; revised November 10, 2000.
The authors are with the Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong.
Publisher Item Identifier S 1045-9227(01)02048-3.