Modified EBP algorithm with instant training of the hidden layer

Bogdan M. Wilamowski
Department of Electrical Engineering
University of Wyoming
Laramie, WY 82071 USA

Abstract **– Several algorithms for training feedforward neural networks including such as the steepest decent EBP and Lavenberg-Marquardt are compared. Various techniques to improve convergence of the EBP are also reviewed. A very fast training algorithm, with instant training of the hidden layer is introduced. For easy problems it has a similar convergence rate as the Lavenberg-Marquardt (LM) method. The algorithm sustain the fast convergence rate also for the cases when the LM algorithm fails and the EBP algorithm has practically unacceptable slow convergence rate.**

I. INTRODUCTION

Although the error backpropagation algorithm (EBP) [13][14][24] was a significant breakthrough in neural network research, it is known as an algorithm with a very poor convergence rate. Many attempts have been made to speed up the EBP algorithm. Commonly known heuristics approaches [4][16][18][19][21] such as momentum [10], variable learning rate [7], or stochastic learning [15], lead only to a slight improvement. Better results were obtained with the artificial enlarging of errors for neurons operating in saturation region [2][8][12][20][25]. More significant improvement was possible by using various second order approaches such as Newton, conjugate gradient, or the Levenberg-Marquardt (LM) method [6]. The LM algorithm is now considered as the most efficient one [6]. It combines the speed of the Newton algorithm with the stability of the steepest decent method. The LM algorithm uses the following formula to calculate weights in subsequent iterations:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}_k^T \mathbf{J}_k + \mu \mathbf{I}\right)^{-1} \mathbf{J}_k^T \mathbf{E} \qquad (1)$$

where **E** is the cumulative (for all patterns) error vector **I** is identity unit matrix, μ is a learning parameter and **J** is Jacobian of m output errors with respect to n weights of neural network. For $\mu = 0$ it becomes the Gauss-Newton method. For very large μ the LM algorithm becomes the steepest decent or the EBP algorithm. The μ parameter is automatically adjusted at each iteration in order to secure convergence. The LM algorithm requires computation of the Jacobian **J** matrix at each iteration step and the inversion of $\mathbf{J}^T\mathbf{J}$ square matrix. The Jacobian matrix is defined as:

$$\mathbf{J} = \begin{matrix} \frac{\partial E_1}{\partial w_1} & \frac{\partial E_1}{\partial w_2} & \frac{\partial E_1}{\partial w_3} & \cdot & \frac{\partial E_1}{\partial w_n} \\ \frac{\partial E_2}{\partial w_1} & \frac{\partial E_2}{\partial w_2} & \frac{\partial E_2}{\partial w_3} & \cdot & \frac{\partial E_2}{\partial w_n} \\ \frac{\partial E_3}{\partial w_1} & \frac{\partial E_3}{\partial w_2} & \frac{\partial E_3}{\partial w_3} & \cdot & \frac{\partial E_3}{\partial w_n} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \frac{\partial E_m}{\partial w_1} & \frac{\partial E_m}{\partial w_2} & \frac{\partial E_m}{\partial w_3} & \cdot & \frac{\partial E_m}{\partial w_n} \end{matrix} \qquad (2)$$

where E_i are cumulative errors for all patterns on i-th output w_j are neural network weights, i = 1 to number of outputs, and j = 1 to number of weights. Unfortunately the Levenberg-Marquardt algorithm often fails when the error surface is unfavorable and initial weights are far from the solution.

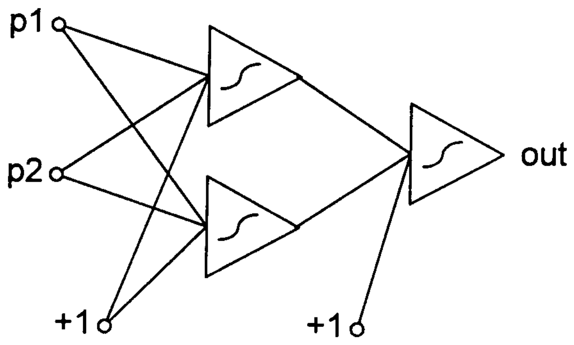

Fig. 1. Simple neural network architecture for "XOR" problem.

The classical "XOR" case shown in Fig. 1 was used for comparison. Depend on initial weights various convergence rates are obtained. Figure 2 shows typical convergence rates for EBP algorithm without momentum and with close to optimum learning rate. The initial weights were chosen using the Nguyen-Widrow weight initialization scheme [11]. Note that the sum of squared errors is relatively large even after 10,000 iterations.

Results of the LM algorithm are shown in Fig.3. Even initial weights were chosen using the Nguyen-Widrow

0-7803-3932-0

weight initialization scheme the Levenberg-Marquardt algorithm fails in 15% to 25% cases, but when converges it usually reach solution in less than 20 steps. Note that the EBP algorithm requires 100 to 1000 times more iteration than the LM algorithm. On other hand the LM algorithm is more computationally intensive at each iteration step.

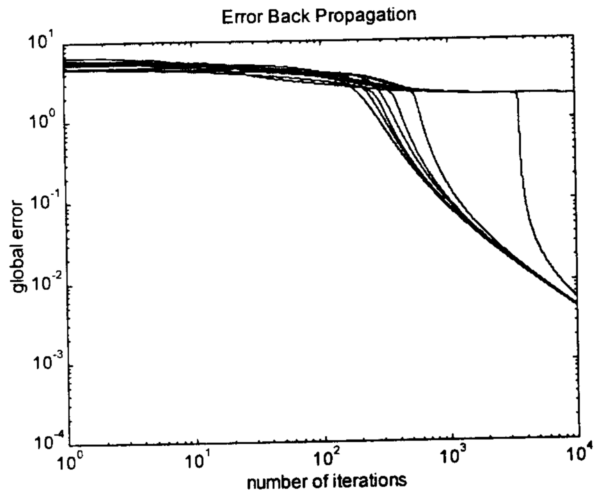

Fig. 2. Sum of squared errors as a function of number of iterations for the "XOR" problem using EBP algorithm with Nguyen-Widrow weight initialization

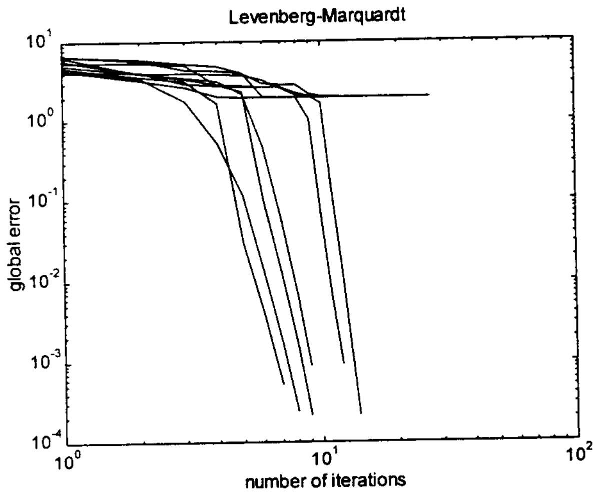

Fig. 3. Sum of squared errors as a function of number of iterations for the "XOR" problem using LM algorithm with Nguyen-Widrow weight initialization. Algorithm failed in 15% to 25% cases

In order to set unfavorable initial weights, very far from the solution, the network was trained for the opposite to desired solution at first then the resulted weights were used as the initial weights for the actual training procedure. When initial weight were chosen purposely very far from the solution the LM algorithm failed in 100% cases and the EBP algorithm converges very slowly as Fig. 4 shows. After 100,000 iterations the error is still not acceptable. A conclusion from this experiment is that with EBP and LM algorithms we can secure the convergence only with a skillful weight initialization.

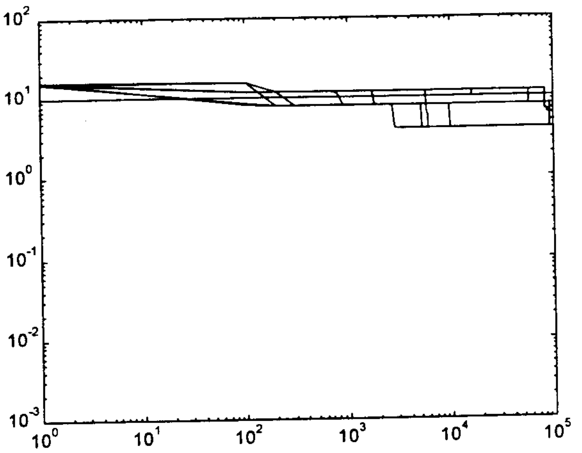

Fig. 4. Sum of squared errors as a function of number of iterations for the"XOR" problem using EBP algorithm with unfavorable weight initialization

A poor convergence, shown in Fig. 4, is not due to local mimima but because of plateaus on the error surface. This problem is also known as "flat spot" problem [2][8][12][20][25]. The prime reason for the plateau formations is a characteristic shape of the sigmoidal activation functions shown in Fig. 5. Note that in the Figure the actual output a significantly differs from the desired output d and the error $ER = d-a$ is very large. At the same time the derivative of the activation function f' at the actual output a is very small, therefore the back propagating error equal to $ER*f'$ is also very small, and the network cannot learn.

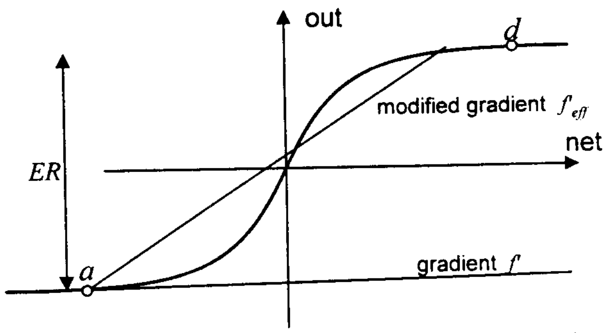

Fig. 5. Activation function with actual and effective gradient. Points a and d represents actual and desired outputs

II. THE ALGORITHM

The success of the proposed algorithm comes from two improvements. In the output layer a modified gradient is used instead of an actual gradient in a similar way as described in [20][25] and weights in the hidden layer are updated using a modified regression algorithm [1].

In the EBP algorithm the back propagating error is proportional to the derivative of the activation function. In the case when this derivative is small, as shown in Fig. 5, the maximally large errors cannot back propagate through the network. This problem can be corrected by introducing an modified gradient instead of the actual one. There are many ways to do this [20][25]. It is important that for small or close to zero error f'_{eff} must be equal *to* f'. In this paper the following simple formula was chosen:

$$f'_{eff} = (1-\alpha)f' + \alpha \tag{3}$$

where:

$$\alpha = (0.01 \sim 0.1)|ER| \tag{4}$$

For small errors α is small and the algorithm behaves as the EBP algorithm, while for large errors the derivatives of the activation function are modified so the error can easily back propagate. Using the modified gradient method as described above, the vector of cumulative errors $\mathbf{E}_l$ (for all patterns) can be found on hidden nodes.

When the modified gradient approach is used the back propagating errors may get very large values and this may lead to an instability of the solution. In order to eliminate this undesired effect the back propagating error is bonded to ±1 using tangent hyperbolic function

$$\Delta E_p = \tanh(E_{hp}) \tag{5}$$

where E_{hp} is the actual back propagating error from the output layer to the hidden layer and ΔE_p is the error used in the computation. Note that for small errors x=tanh(x) and

$$\Delta E_p \approx E_{hp} \tag{6}$$

Learning process for the first layer can be significantly speed up using the modified regression approach. The error function on a hidden neuron with n weights and a given input pattern p is a function of the first layer weights.

$$E_p = f\left(\sum w_1 p_{p1} + w_2 p_{p2} + \cdots + w_l p_{pl}\right) - D_p \tag{7}$$

where D_p is the desired output. The error increment ΔE_p can then be approximated by the first two terms of the linear approximation around a given point:

$$\Delta E_p = \frac{dE_p}{dw_1}\Delta w_1 + \frac{dE_p}{dw_2}\Delta w_2 + \cdots\cdots \frac{dE_p}{dw_n}\Delta w_n \tag{8}$$

From (7)

$$\frac{dE_p}{dw_i} = \frac{dE_p}{df}\frac{df}{dw_i} = f'_p p_{pi} \tag{9}$$

by inserting (9) into (8)

$$\Delta E_p = -f'_p\left(p_{p1}\Delta w_1 + p_{p2}\Delta w_2 + \ldots + p_{pn}\Delta w_l\right) \tag{10}$$

therefore, using matrix notation, for all input patterns:

$$\begin{bmatrix} p_{11} & p_{12} & p_{12} & \cdots & p_{1l} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2l} \\ \vdots & \vdots & \vdots & & \vdots \\ p_{p1} & p_{p2} & p_{p3} & \cdots & p_{pl} \\ \vdots & \vdots & \vdots & & \vdots \\ p_{P1} & p_{P2} & p_{P3} & \cdots & p_{Pl} \end{bmatrix} \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_l \end{bmatrix} = \begin{bmatrix} \frac{\Delta E_1}{f'_1} \\ \frac{\Delta E_2}{f'_2} \\ \vdots \\ \frac{\Delta E_p}{f'_l} \\ \vdots \\ \frac{\Delta E_P}{f'_l} \end{bmatrix} \tag{11}$$

The input matrix **P** [l·P] in (8) is rectangular. Where l is the number of inputs and P is the number of training patterns. When l is lager than P the problem is trivial and has an infinite number of solutions. In the case when l is smaller than P the pseudo inverse of the matrix is found in least mean square sense:

$$\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T \tag{12}$$

Since **P** is the array of input patterns and it is constant during the learning procedure, therefore pseudo inversion of input patterns **P** must be done only once. At each iteration step only new values for ΔE and f'_{eff} must be computed. The weights change Δw is calculated as

$$\Delta w = P^{+} \frac{\Delta E}{f'_{eff}} \tag{13}$$

where P^{+} is constant during all training procedure. When this approach is used for one layer neural network the solution is usually reached in three to six iterations. In the case of two layer networks considered herein the modified EBP algorithm is used in the output layer, therefore, more iterations are required. Usually a satisfactory solution is reached in less than 20 iterations.

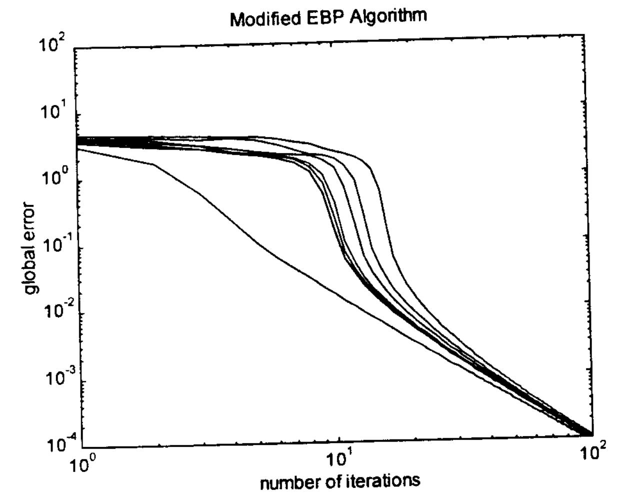

Fig. 6. Sum of squared errors as a function of number of iterations for the"XOR" problem using modified EBP algorithm with Nguyen-Widrow weight initialization

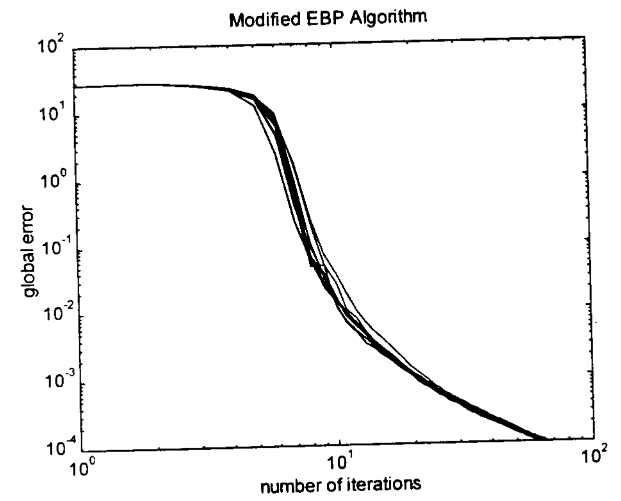

Fig. 7. Sum of squared errors as a function of number of iterations for the"XOR" problem using modified EBP algorithm with unfavorable weight initialization

III. EXAMPLES

The proposed algorithm was verified on several examples with different sizes and different roughness of the error surfaces. Several bench mark problems XOR, parity 4, parity 8 and bottleneck problems, were used. Some practical problems as speaker identification [22] (with twenty inputs) or recognition of noisy letters [9] (with 56 inputs) were also used for algorithm verification. In smooth and easy cases the results are similar to those obtained with the LM algorithm. In difficult cases, with far from solution initial weights, where the LM algorithm always fails and the EBP converges very slowly, the presented algorithm converged rapidly to the solution. Figures 6 and 7 show results of the algorithm for the same "XOR" case as it was shown in Figures 2, 3, and 4.

IV. CONCLUSION

A very efficient algorithm for training two layer neural networks was presented and verified with several examples. The algorithm is advantageous to other commonly used algorithms. It converges very rapidly for the cases when the EBP and LM algorithms fail. Convergence rate is almost independent on the choice of initial weights.

In contrary to the LM algorithm the matrix inversion must be done only once and also much smaller matrix must be inverted. The size of the matrix corresponds to the number of inputs, while in the LM algorithm the size of the matrix corresponds to the number of weights in the neural network. The large size of this matrix is a significant drawback of the Levenberg-Marquardt algorithm is that, due to memory limitation it can be only used for relative small neural network [6]. The proposed algorithm does not have this limitation since the matrix is equal only to the number of inputs and the matrix inversion must be done only once.

V. REFERENCES

[1] Andersen, Thomas J. and B.M. Wilamowski, "A. Modified Regression Algorithm for Fast One Layer Neural Network Training", World Congress of Neural Networks, vol. 1, pp. 687-690, Washington DC, USA, July 17-21, 1995.

[2] Balakrishnan, K. & Honavar, V. (1992). Improving convergence of back propagation by handling flat-spots in the output layer. *Proceedings of Second International Conference on Artificial Neural Networks*, Brighton, U.K. Barmann F. & Biegler-Konig, F. (1992). On class of efficient learning algorithms for neural networks. *Neural Networks*, 5, 139-144.

[3] Battiti R., "First- and second-order methods for learning: between steepest descent and Newton's method, Neural Computation, vol. **4**, no. 2, pp. 141-166, 1992.

[4] Bello, M. G. (1992). Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptron networks. *IEEE Trans. on Neural Networks*, **3**, 864-875.

[5] Charalambous C., "Conjugate gradient algorithm for efficient training of artificial neural networks," IEE Proceedings, vol. **139**, no. 3, pp. 301-310, 1992.

[6] Hagan M. T. and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," IEEE Transactions on Neural Networks, vol. **5**, no. 6, pp. 989-993, 1994.

[7] Jacobs R. A., "Increased rates of convergence through learning rate adaptation," Neural Networks, vol. **1**, no.4, pp. 295-308, 1988.

[8] Krogh, A., Thorbergsson, G. I. & Hertz, J. A. (1989). A cost function for internal representations. In D. Touretzky (Eds.), *Advances in neural information processing systems II* (pp. 733-740). San Mateo, Ca.

[9] McInroy John E. and Bogdan M. Wilamowski "Bipolar Pattern Association Using A Recurrent Winner Take All Network" *International Conference on Neural Networks - ICNN 1997* vol. **2**, pp. 1231-1234.

[10] Miniani, A. A. & Williams, R. D. (1990). Acceleration of back-propagation through learning rate and momentum adaptation. *Proceedings of International Joint Conference on Neural Networks*, San Diego, CA, **1**, 676-679.

[11] Nguyen D., B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of adaptive weights," in Proc. IJCNN, vol. **3**, pp. 21-26, July 1990.

[12] Parekh, R., Balakrishnan, K. & Honavar, V. (1992). An empirical comparison of flat-spot elimination techniques in back-propagation networks. *Proceedings of Third Workshop on Neural Networks - WNN'92*, Auburn, pp. 55-60.

[13] Rumelhart D. E., G. E. Hinton, R. J. Williams, Learning internal representations by error propagation. In Parallel Distributed Processing, vol **1**, pp. 318-362. Cambridge, MA: MIT Press.

[14] Rumenhart D. E., G. E. Hinton and R. J. Wiliams, "Learning representations by back-propagating errors" Nature, vol. **323**, pp. 533-536, 1986

[15] Salvetti A., B. Wilamowski, "Introducing Stochastic Process within the Backpropagation Algorithm for Improved Convergence" presented at *ANNIE'94 - Artificial Neural Networks in Engineering*, St. Louis, Missouri, USA, November 13-16, 1994; also in *Intelligent Engineering Systems Through Artificial Neural Networks* vol **4**, pp. 205-209, ed. C. H. Dagli, B. R. Fernandez, J. Gosh, R.T. S. Kumara, ASME PRESS, New York 1994.

[16] Samad, T. (1990). Back-propagation improvements based on heuristic arguments. *Proceedings of International Joint Conference on Neural Networks*, Washington, **1**, 565-568.

[17] Shah, S. & Palmieri, F. (1990). MEKA - A fast, local algorithm for training feedforward neural networks. *Proceedings of International Joint Conference on Neural Networks*, San Diego, CA, **3**, 41-46.

[18] Solla, S. A., Levin, E. & Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Systems*, **2**, 625-639.

[19] Sperduti, A. & Starita, A. (1993). Speed up learning and network optimization with extended back-propagation. *Neural Networks*, **6**, 365-383.

[20] Torvik, L. and B. M. Wilamowski, "Modification of the Backpropagation Algorithm for Faster Convergence", presented at *1993 International Simulation Technology Multiconference* November 7-10, San Francisco; also in proceedings of Workshop on Neural Networks WNN93 pp. 191-194, 1993

[21] Van Ooten A. & Nienhuis B. (1992). Improving the convergence of the back-propagation algorithm. *Neural Networks*, **5**, 465-471.

[22] Vieira Karina, Bogdan M. Wilamowski, and Robert Speaker Identification Based on a Modified Kohonen Network" *International Conference on Neural Networks - ICNN 1997* vol. **4**, pp. 2103-2106.

[23] Wartous, R. L. (1987). Learning algorithms for connectionist networks: applied gradient methods of non-linear optimization. *Proceedings of Conference on Neural Networks*, San Diego, CA, **2**, 619-627.

[24] Werbos, P. J. (1988). Back-propagation: Past and future. *Proceeding of International Conference on Neural Networks*, San Diego, CA, **1**, 343-354.

[25] Wilamowski, B.M. and L. Torvik, "Modification of Gradient Computation in the Back-Propagation Algorithm", presented at *ANNIE'93 - Artificial Neural Networks in Engineering*, St. Louis, Missouri, November 14-17, 1993; also in *Intelligent Engineering Systems Through Artificial Neural Networks* vol. **3**, pp. 175-180, ed. C. H. Dagli, L. I. Burke, B. R. Fernandez, J. Gosh, R.T., ASME PRESS, New York 1993.

Proceedings of the

IECON '97

23rd International Conference on
Industrial Electronics,
Control, and Instrumentation

Volume 3 of 4

97CH36066