Efficient Algorithm for Training Neural Networks with one Hidden Layer

Bogdan M. Wilamowski, Yixin Chen,
EE Department, University of Wyoming, Laramie, WY 82071
and Aleksander Malinowski
ECET Department, Bradley University, Peoria, IL 61625
wilam@ieee.org, dillion@uwyo.edu, olekmail@ieee.org

Abstract -- *Efficient second order algorithm for training feedforward neural networks is presented. The algorithm has a similar convergence rate as the Lavenberg-Marquardt (LM) method and it is less computationally intensive and requires less memory. This is especially important for large neural networks where the LM algorithm becomes impractical. Algorithm was verified with several examples.*

1. Introduction

The error backpropagation algorithm (EBP) [13][14][23] was a significant breakthrough in neural network research, but it is also 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 the saturation region [2][8][12][20]. More significant improvement was possible by using various second order approaches such as Newton, conjugate gradient, or the Levenberg-Marquardt (LM) method [1][3][5][6][17]. The LM algorithm is now considered as the most efficient[6]. It combines the speed of the Newton algorithm with the stability of the steepest decent method.

The main disadvantage of the LM algorithm is its demand for memory to operate with large Jacobians and a necessity of inverting large matrixes. The rank of matrixes to be inverted is equal to the number of weights in the system. Such large matrixes must be inverted at each iteration step and this results in large computation time.

2. Levenberg-Marquardt Algorithm (LM)

For LM algorithm, the performance index to be optimized is defined as

$$F(\mathbf{w})=\sum_{p=1}^{P}\left[\sum_{k=1}^{K}\left(d_{kp}-o_{kp}\right)^2\right] \tag{1}$$

where $\mathbf{w}=[w_1 \quad w_2 \quad \dots \quad w_N]^T$ consists of all weights of the network, d_{kp} is the desired value of the k^{th} output and the p^{th} pattern, o_{kp} is the actual value of the k^{th} output and the p^{th} pattern, N is the number of the weights, P is the number of patterns, and K is the number of the network outputs.

Equation (1) can be written as

$$F(\mathbf{w})=\mathbf{E}^T\mathbf{E} \tag{2}$$

where

$$\mathbf{E}=[e_{11} \quad \dots \quad e_{K1} \quad e_{12} \quad \dots \quad e_{K2} \quad \dots \quad e_{1P} \quad \dots \quad e_{KP}]^T$$

$$e_{kp}=d_{kp}-o_{kp}, \quad k=1,\cdots,K, \quad p=1,\cdots,P$$

where $\mathbf{E}$ is the cumulative error vector (for all patterns). From equation (2) the Jacobian matrix is defined as

$$\mathbf{J}=\begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_N} \\ \frac{\partial e_{21}}{\partial w_1} & \frac{\partial e_{21}}{\partial w_2} & \cdots & \frac{\partial e_{21}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{K1}}{\partial w_1} & \frac{\partial e_{K1}}{\partial w_2} & \cdots & \frac{\partial e_{K1}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{1P}}{\partial w_1} & \frac{\partial e_{1P}}{\partial w_2} & \cdots & \frac{\partial e_{1P}}{\partial w_N} \\ \frac{\partial e_{2P}}{\partial w_1} & \frac{\partial e_{2P}}{\partial w_2} & \cdots & \frac{\partial e_{2P}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{KP}}{\partial w_1} & \frac{\partial e_{KP}}{\partial w_2} & \cdots & \frac{\partial e_{KP}}{\partial w_N} \end{bmatrix} \tag{3}$$

and the weights are calculated using the following equation

$$\mathbf{w}_{t+1}=\mathbf{w}_t-\left(\mathbf{J}_t^T\mathbf{J}_t+\mu_t\mathbf{I}\right)^{-1}\mathbf{J}_t^T\mathbf{E}_t \tag{4}$$

where **I** is identity unit matrix, μ is a learning parameter and **J** is Jacobian of m output errors with respect to n weights of the neural network. For $\mu = 0$ it becomes the Gauss-Newton method. For very large μ the LM algorithm becomes the

0-7803-5529-6/99/$10.00 ©1999 IEEE

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 $\mathbf{J}$ matrix at each iteration step and the inversion of $\mathbf{J}^T\mathbf{J}$ square matrix. Note that in the LM algorithm an N by N matrix must be inverted in every iteration. This is the reason why for large size neural networks the LM algorithm is not practical. We are proposing another method that provides a similar performance, while lacks the inconveniences of LM, and is more stable.

3. Modification of the LM Algorithm

Instead of the performing index given by (1), the following new performing index is introduced

$$F(\mathbf{w}) = \sum_{k=1}^{K}\left[\sum_{p=1}^{P}\left(d_{kp} - o_{kp}\right)^2\right]^2 \tag{5}$$

This form of the index, which represents a global error, will later lead to a significant reduction of the size of a matrix to be inverted at each iteration step. Equation (5) can be also written as:

$$F(\mathbf{w}) = \hat{\mathbf{E}}^T\hat{\mathbf{E}} \tag{6}$$

where

$$\hat{\mathbf{E}} = \left[\hat{e}_1 \quad \hat{e}_2 \quad \cdots \quad \hat{e}_K\right]^T$$

$$\hat{e}_k = \sum_{p=1}^{P}\left(d_{kp} - o_{kp}\right)^2, \quad k = 1, \cdots K$$

Now the modified Jacobian matrix $\hat{\mathbf{J}}_t$ can be defined as

$$\hat{\mathbf{J}} = \begin{bmatrix} \frac{\partial \hat{e}_1}{\partial w_1} & \frac{\partial \hat{e}_1}{\partial w_2} & \cdots & \frac{\partial \hat{e}_1}{\partial w_N} \\ \frac{\partial \hat{e}_2}{\partial w_1} & \frac{\partial \hat{e}_2}{\partial w_2} & \cdots & \frac{\partial \hat{e}_2}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial \hat{e}_K}{\partial w_1} & \frac{\partial \hat{e}_K}{\partial w_2} & \cdots & \frac{\partial \hat{e}_K}{\partial w_N} \end{bmatrix} \tag{7}$$

and equation (4) can be written using the modified Jacobian matrix $\hat{\mathbf{J}}_t$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \left(\hat{\mathbf{J}}_t^T\hat{\mathbf{J}}_t + \mu_t\mathbf{I}\right)^{-1}\hat{\mathbf{J}}_t^T\hat{\mathbf{E}}_t \tag{8}$$

Note $\hat{\mathbf{J}}_t$ is a K by N matrix, and it still leads to a necessity of inverting an N by N matrix. Where N is the number of weights. This problem can be now further simplified using the Matrix Inversion Lemma which states that if a matrix $\mathbf{A}$ satisfies

$$\mathbf{A} = \mathbf{B}^{-1} + \mathbf{C}\mathbf{D}^{-1}\mathbf{C}^T \tag{9}$$

then

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{B}\mathbf{C}\left(\mathbf{D} + \mathbf{C}^T\mathbf{B}\mathbf{C}\right)^{-1}\mathbf{C}^T\mathbf{B} \tag{10}$$

Let

$$\mathbf{A} = \mathbf{J}_t^T\mathbf{J}_t + \mu_t\mathbf{I} \tag{11}$$

$$\mathbf{B} = \frac{1}{\mu_t}\mathbf{I} \tag{12}$$

$$\mathbf{C} = \mathbf{J}_t^T \tag{13}$$

$$\mathbf{D} = \mathbf{I} \tag{14}$$

Substituting equations (11), (12), (13), and (14) into equation (10), one can obtain

$$\left(\hat{\mathbf{J}}_t^T\hat{\mathbf{J}}_t + \mu_t\mathbf{I}\right)^{-1} = \frac{1}{\mu_t}\mathbf{I} - \frac{1}{\mu_t^2}\hat{\mathbf{J}}_t^T\left(\mathbf{I} + \frac{1}{\mu_t}\hat{\mathbf{J}}_t\hat{\mathbf{J}}_t^T\right)^{-1}\hat{\mathbf{J}}_t \tag{15}$$

Note that in the right side of equation (15), the matrix to be inverted is of size K by K. In every application, N, which is number of weights, is much greater than K, which is number of outputs. Thus, using equation (15) the intensity of computation can be significantly reduced.

By inserting equation (15) into equation (8) one may have

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \left[\frac{1}{\mu_t}\mathbf{I} - \frac{1}{\mu_t^2}\hat{\mathbf{J}}_t^T\left(\mathbf{I} + \frac{1}{\mu_t}\hat{\mathbf{J}}_t\hat{\mathbf{J}}_t^T\right)^{-1}\mathbf{J}_t\right]\hat{\mathbf{J}}_t^T\hat{\mathbf{E}}_t \tag{16}$$

For single output networks, equation (10) becomes

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{\mu_t}\left[\mathbf{I} - \frac{\hat{\mathbf{J}}_t^T\hat{\mathbf{J}}_t}{\mu_t + \hat{\mathbf{J}}_t\hat{\mathbf{J}}_t^T}\right]\hat{\mathbf{J}}_t^T\hat{\mathbf{E}}_t \tag{17}$$

Note that in equation (17) matrix inversion is not required at all. The equation is useful, because any feedforward network with one hidden layer and K outputs can be decoupled to a K single output network.

With the proposed modifications much larger neural networks can be trained than in the case when the LM algorithm.

4. Examples

The proposed algorithm was verified on several examples with different sizes and different roughness of the error surfaces. Several benchmark problems XOR, parity 3, parity 4, and letter recognition [9] were used.

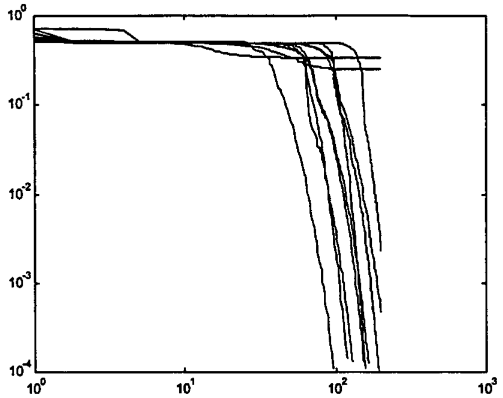

Figure 1. XOR problem with 2 hidden neurons.

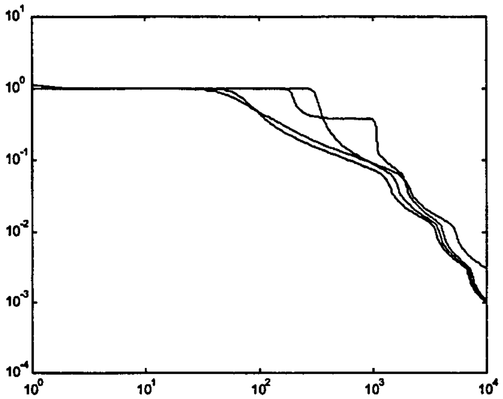

Figure 2. Parity 3 problem with 2 hidden neurons.

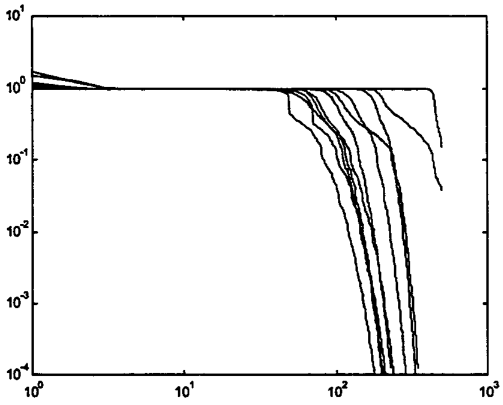

Figure 3. Parity 3 problem with 3 hidden neurons.

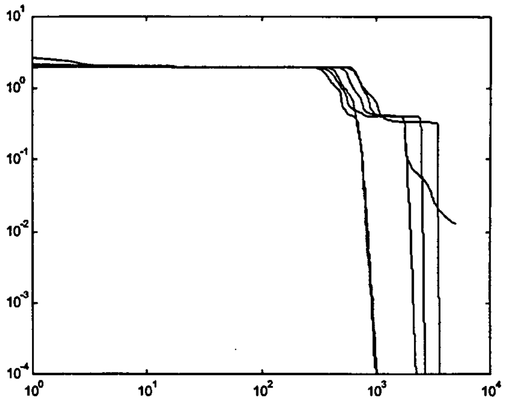

Figure 4. Parity 4 problem with 6 hidden neurons.

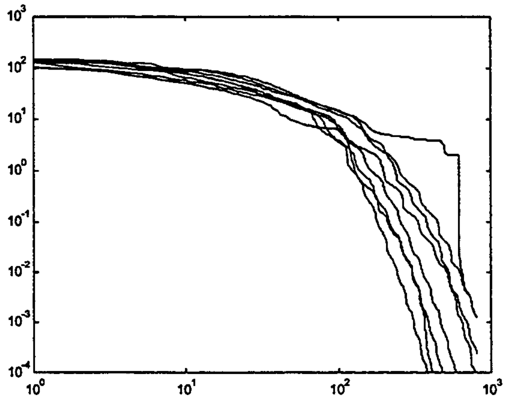

Figure 5. Character Learning with 10 hidden neurons.

5. Conclusion

A fast and efficient training algorithm for feedforward neural networks with one hidden layer was developed and tested on several examples. An even number of required iterations is slightly larger than in the LM algorithm, yet it is less computationally intensive and less memory demanding.

The size of the matrix corresponds to the number of outputs, 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; due to memory limitations it can be only used for relatively small neural networks. 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.

6. 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] 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.

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