An Algorithm for Fast Convergence in Training Neural Networks

Bogdan M. Wilamowski
University of Idaho
Graduate Center at Boise
800 Park Blvd., Boise, U.S.A.
wilam@ieee.org

Serdar Iplikci
Bogazici University
Electrical and Electronic Engineering Department
Bebek, 80815, Istanbul, Turkey,
kaynak@boun.edu.tr

Okyay Kaynak
Bogazici University
Electrical and Electronic Engineering Department
Bebek, 80815, Istanbul, Turkey
kaynak@boun.edu.tr

M. Önder Efe
Carnegie Mellon University
Electrical and Computer Engineering Department
Pittsburgh, PA 15213-3890, U.S.A.
efemond@andrew.cmu.edu

Abstract

In this work, two modifications on Levenberg-Marquardt algorithm for feedforward neural networks are studied. One modification is made on performance index, while the other one is on calculating gradient information. The modified algorithm gives a better convergence rate compared to the standard Levenberg-Marquard (LM) method and is less computationally intensive and requires less memory. The performance of the algorithm has been checked on several example problems.

1 Introduction

Although the Error Backpropagation algorithm (EBP) [1][2][3] has been a significant milestone in neural network research area of interest, it has been known as an algorithm with a very poor convergence rate. Many attempts have been made to speed up the EBP algorithm. Commonly known heuristic approaches [4][5][6][7][8] such as momentum [9], variable learning rate [10], or stochastic learning [11] lead only to a slight improvement. Better results have been obtained with the artificial enlarging of errors for neurons operating in the saturation region [12][13][14][15]. A significant improvement on realization performance can be observed by using various second order approaches namely Newton's method, conjugate gradient's, or the Levenberg-Marquardt (LM) optimization technique [16][17][18][19] [20]. Among the mentioned methods, the LM algorithm is widely accepted as the most efficient one in the sense of realization accuracy [19]. It gives a good compromise between the speed of the Newton algorithm and the stability of the steepest descent method, and consequently it constitutes a good transition between these methods.

The demand for memory to operate with large Jacobians and a necessity of inverting large matrices are the major disadvantages of the LM algorithm. The rank of the matrix to be inverted at each iteration is equal to the number of adjustable parameters in the system. As the dimensionality of the network increases, it should be clear that the training would entail costly hardware due to the exponential growth in the computational complexity.

This paper is organized as follows: the second section describes the Levenberg-Marquardt algorithm. In the third section, the proposed form of the modification on performance index is introduced. Next section focuses on the modification of the gradient computation. Exemplar cases are discussed in the forth section. Finally, conclusions constitute the last part of the paper.

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 $\mathbf{w} = [w_1 \quad w_2 \quad \ldots \quad w_N]^T$ 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

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

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

where 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

$$\mathbf{w_{t+1}} = \mathbf{w_t} - \left(\mathbf{J_t^T J_t} + \mu\mathbf{I}\right)^{-1}\mathbf{J_t^T E_t} \tag{4}$$

where $\mathbf{I}$ is identity unit matrix, μ is a learning parameter and $\mathbf{J}$ 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 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, the dimension of which is N×N. This is the reason why for large size neural networks the LM algorithm is not practical. In this paper, a novel method is proposed that provides a similar performance, while lacks the inconveniences of LM, and, furthermore, is more stable.

3 Modification of the performance index

Assume that the performance index of (1) is changed to the one given below. Note that still the continuity requirements are preserved and that the new measure can well be considered as a measure of similarity between the desired and the produced patterns.

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

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

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

where $\hat{\mathbf{E}} = \begin{bmatrix}\hat{e}_1 & \hat{e}_2 & \ldots & \hat{e}_K\end{bmatrix}^T$ and $\hat{e}_k = \sum_{p=1}^{P}\left(d_{kp} - o_{kp}\right)^2$ with k=1,…,K. Now the modified Jacobian matrix $\hat{\mathbf{J}}_t$ can be defined as

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

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

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

It should be noted that although $\hat{\mathbf{J}}_t$ is now a *K* by *N* matrix, and there still a necessity of inverting an *N* by *N* matrix, where *N* is the number of adjustable parameters. 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

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

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

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

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

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

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

Note that in the right side of equation (15), the matrix to be inverted is of size *K* by *K*. In most neural network applications, *N*, which is the number of weights, is much greater than *K*, which is the number of outputs. This means that by use of equation (5) instead of equation (1) the computational complexity of the weight adaptation problem is significantly reduced.

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

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

For single output networks, equation (10) becomes

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

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.

4 Modification of gradient computation

The major disadvantage of back-propagation algorithm, commonly employed for training of multilayer networks, is slow asymptotic convergence rate. For the sigmoidal bipolar activation function given by (18)

$$f(net) = \frac{2}{1+\exp(-\lambda\ net)} - 1 \qquad (18)$$

The gradient (slope) is computed as a derivative of (18)

$$g_1 = \frac{2\lambda\exp(-\lambda\ net)}{1+\exp(-\lambda\ net)} = 0.5\lambda(1 - f^2) \qquad (19)$$

The activation function and its gradient are illustrated in Figure 1.

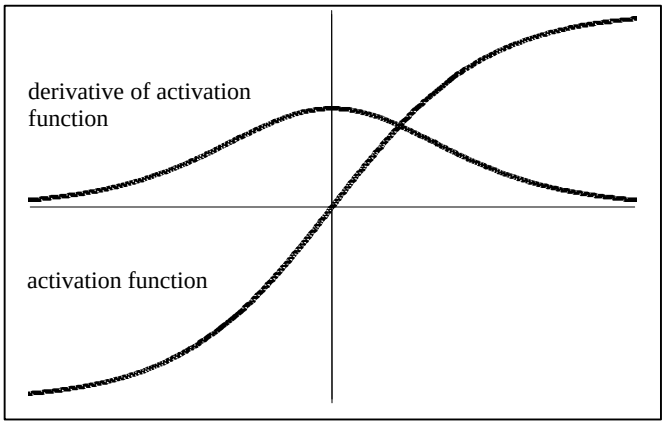

Figure 1: Standard sigmoidal function for bipolar neurons and its derivative

In the error backpropagation algorithm, the weight updates are proportional to the error propagating from the output through the derivatives of activation function and through the weights. This is a consequence of using the steepest descent method for calculating the weight adjustments. Convergence properties of the learning process can be improved by changing how the error propagates back through the network. It is proposed in this paper that for purpose of error propagation, the slope (gradient) of the activation function is calculated as the slope of the line connecting the output value with the desired value, rather than the derivative of the activation function at the output value. This is illustrated in Figure 2.

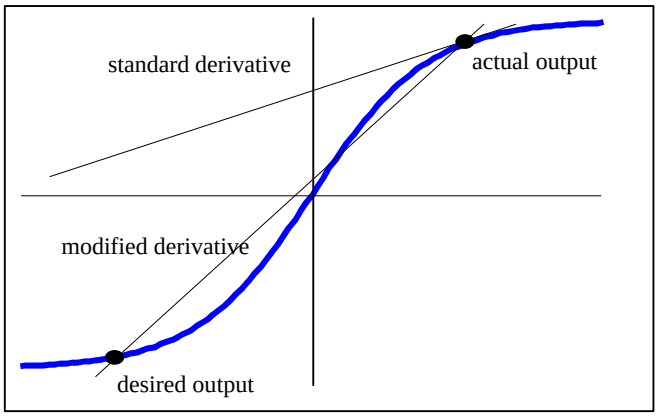

Figure 2: Illustration of the modified derivative computation using slope of the line connecting the points of actual output and desired output

Note that if the output is close to the desired value, the calculated slope corresponds to the derivative of the activation function, and the algorithm is identical to the standard backpropagation formula. Therefore, the "derivative" is calculated in a different manner only for large errors when the classical approach significantly limits error propagation.

5 Examples

The proposed algorithm has been experimented on several problems with different network topologies and different roughness of the error surfaces. Several benchmark problems namely XOR, parity-3, and parity-4 have been taken into consideration. Table I summarizes the performance of the modified algorithm compared to standard LM algorithm. Using the slope instead of the derivative yields better performance in the number of average iterations for meeting the prescribed convergence criterion. The results have been given in Figures 3-6.

6 Conclusions

A fast and efficient training algorithm for feedforward neural networks with one hidden layer has been presented and tested on several examples. Since in the LM algorithm, the size of the matrix corresponds to the number of weights in the neural network, while the size of the matrix, in the modified algorithm, corresponds to the number of outputs, the modified algorithm requires less memory than that for the standard LM algorithm. Moreover, an improvement in the number of average iterations for convergence has been obtained by introducing a different tool for gradient computation.

Acknowledgments

The IEEE Neural Network Council, Student Summer Research Support Program supports this work.

Table I. Comparison of methods

		Derivative	Slope
XOR	# of hidden neurons	*2*	*2*
	Average # of iterations	*50*	*36*
Parity-3	# of hidden neurons	*2*	*2*
	Average # of iterations	*64*	*46*
	# of hidden neurons	*3*	*3*
	Average # of iterations	*45*	*35*
Parity-4	# of hidden neurons	*6*	*6*
	Average # of iterations	*175*	*134*

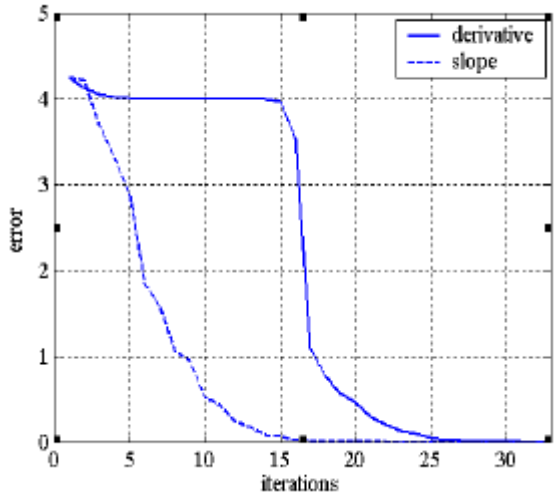

Figure 3: XOR problem with 2 hidden neurons

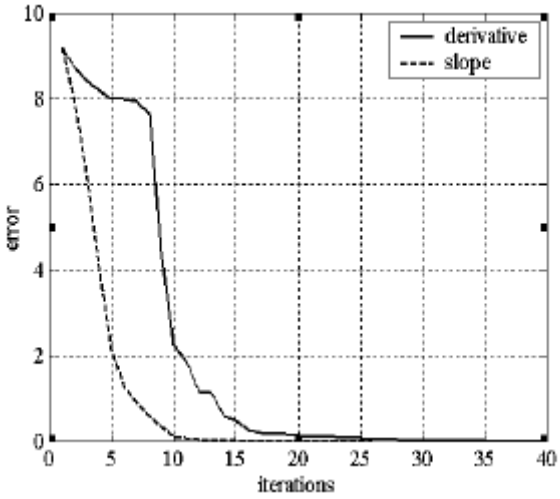

Figure 4: Parity 3 problem with 2 hidden neurons

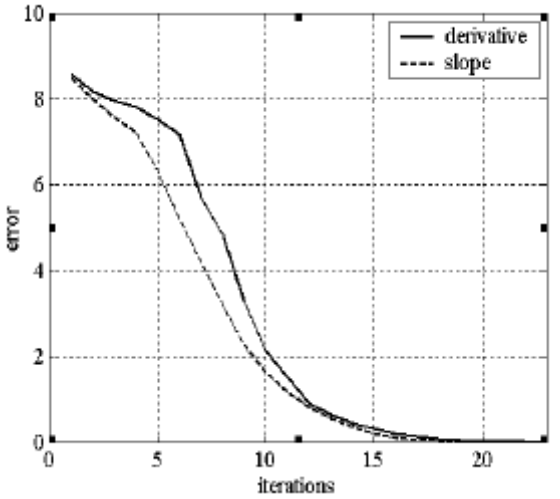

Figure 5: Parity 3 problem with 3 hidden neurons

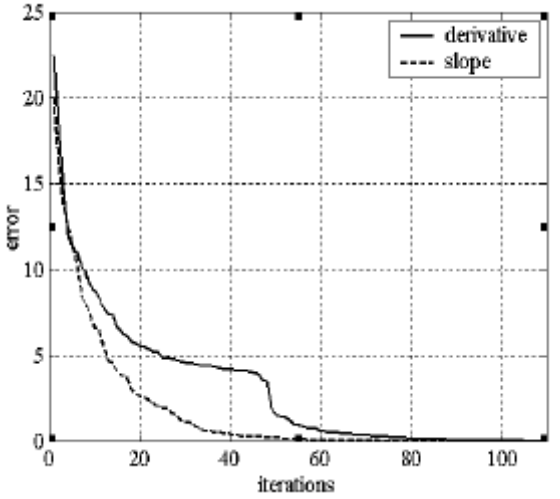

Figure 6: Parity 4 problem with 6 hidden neurons

References

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

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

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

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

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

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

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

[9] Miniani, A. A. and 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.

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

[11] Salvetti, A. and Wilamowski, B. M., "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.

[12] Balakrishnan, K. and 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. and Biegler-Konig, F. (1992). On class of efficient learning algorithms for neural networks. *Neural Networks*, 5**,** 139-144.

[13] Krogh, A., Thorbergsson, G. I. and 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.

[14] Parekh, R., Balakrishnan, K. and 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.

[15] Torvik, L. and Wilamowski, B. M., "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.

[16] Andersen, T. J. and Wilamowski, B.M. "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.

[17] 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.

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

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

[20] Shah, S. and 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.